Lab 1

Stack Implementation & Expression Evaluation

A stack is a container of objects that are inserted and removed according to the last in, first-out (LIFO) principle.
• Inserting an item is known as "pushing" onto the stack.
• Removing an item is known as "Popping" from the stack
So there are 2 main operations in stack **push** and **pop**

Part 1:
It's required to implement Stack (Array Based or Linkedlist Based) with the following function: -

1. Initialize
Prototype → `Stack* initialize();`
It initializes stack so that there are no elements inserted.

2. Pop
Prototype → `Item pop (Stack *s);`
It removes the last inserted element in the stack and return it.

3. Push
Prototype → `void push(Stack *s,Item value);`
It inserts element at the top of the stack.

4. Top
Prototype → `Item top (Stack *s);`
It returns the last inserted element in the stack without removing it.

5. isEmpty
Prototype →`int isEmpty(Stack *s);`
It returns 1 if stack is empty or 0 otherwise.

Part 2:

Write a C function that takes a infix expression as input and convert it to postfix

Function prototype →

```
void infixTopostfix(char *infix, char* postfix);
```

Note that infix input is the infix expression and postfix is an empty array and will be filled with postfix expression

Part 3:

Write a C function that takes a postfix expression as input and shows the value of the value of the expression as output.

The input will be a postfix (not infix) and you have to use your stack implementation to evaluate the expression.

Function prototype → `float evaluatePostfix(char* postfix);`

Part 4:

The main should take a string as input from user, convert it to postfix notation using infixToPostfix(), and then call evaluatePostfix().

Cases that must be handled in the program

- Single digit numbers
- Multi digit numbers
- Brackets
- Floating point numbers
- Negative numbers

Examples

Input (Infix): 1 + 2 * 4 + 3

Output (Postfix): 1 2 4 * + 3 +

Value: 12.0

Input (Infix): (1 + 2) * 4 + 3

Output (Postfix): 1 2 + 4 * 3 +

Value: 15.0

Input (Infix): 10 + 3 * 5 / (16 – 4)

Output (Postfix): 10 3 5 * 16 4 - / +

Value: 11.25

Input (Infix): 2 + 3 * 4

Output (Postfix): 2 3 4 * +

Value: 14.0

Input (Infix): 2 + (-2.5 + 3.14) * (-5.4 + 8.1) ^ (-0.5)

Output (Postfix): 2 -2.5 3.14 + -5.4 8.1 + -0.5 ^ * +

Value: 2.389492

Notes

- Implement your algorithms using c/c++ programming language.
- You must follow the provided template.
- You should work in groups **of 2 members**
- Discussion will have higher weight than implementation, so you should understand your implementation well to get discussion marks.
- It's better to deliver nothing than delivering a copy
- Copied assignments will be severely penalized.