Alexandria University
Faculty of Engineering
Specialized Scientific Programs
Computer & Communication Program
Spring 2025

Data Structure (1)
Course Code: CSE127
Lecturer: Prof. Dr. Nagia M. Ghanem
Dr. Sami Hafez

Expression Evaluation Assignment

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 using LinkedList with the following functions:

1. **Initialize**
   Prototype → **`Stack* initialize ();`**
   It initializes the stack so that there are no elements inserted.
2. **Pop**
   Prototype → **`float pop (Stack *s);`**
   It removes the last inserted element in the stack and returns it.
3. **Push**
   Prototype → **`void push (Stack *s, float value);`**
   It inserts elements at the top of the stack.
4. **peek**
   Prototype → **`float peek (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 the stack is empty or 0 otherwise.

Part 2:

Write a C function that takes an infix expression as input and converts it to postfix.

Function prototype →

```
char* infixTopostfix(char *infix);
```

Note that infix input is the infix expression and function should return the postfix expression.

Part 3:

Write a C function that takes a postfix expression as input and shows 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 the 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
- You should handle the power operation ^ and it has higher priority than * / %

Examples

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

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

Value: 12.0

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

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

Value: 15.0

3- Input (Infix): 10 + 3 * 5 / (16 – 4)

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

Value: 11.25

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

Output (Postfix): 2 3 4 * +

Value: 14.0

5- 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:

- You should work in groups **of 2 members.**
- You can consider that the input will be separated by space so that you can tokenize the input according to that.