CS152: Data Structure

Week 7 Knowledge Check - Stack Applications

Write the full name of all collaborators inside this box

Discuss these questions and write your answers in the space provided below it.

1. Following the example given in the slides, evaluate by hand the following postfix expressions: Show your working in the table below each expression.

a. 10 5 4 +*
b. 10 5 * 6 -
c. 22 2 4 * /
d. 33 6 + 3 4 / +

Below is a step-by-step evaluation of each postfix expression using a stack. Each row shows the stack state after processing each token.

a. 10 5 4 + *

Step	Stack State	Operation / Explanation
1	10	Push 10
2	10, 5	Push 5
3	10, 5, 4	Push 4
4	10, 9	Pop 5 and 4, compute 5 + 4 = 9, push 9
5	90	Pop 10 and 9, compute 10 * 9 = 90, push 90

b. 10 5 * 6 -

Step	Stack State	Operation / Explanation
1	10	Push 10
2	10, 5	Push 5
3	50	Pop 10 and 5, compute 10 * 5 = 50, push 50
4	50, 6	Push 6
5	44	Pop 50 and 6, compute 50 - 6 = 44, push 44

c. 22 2 4 * /

Step	Stack State	Operation / Explanation
1	22	Push 22
2	22, 2	Push 2
3	22, 2, 4	Push 4
4	22, 8	Pop 2 and 4, compute 2 * 4 = 8, push 8
5	2.75	Pop 22 and 8, compute 22 / 8 = 2.75, push 2.75

d. 33 6 + 3 4 / +

Step	Stack State	Operation / Explanation
1	33	Push 33
2	33, 6	Push 6
3	39	Pop 33 and 6, compute 33 + 6 = 39, push 39
4	39, 3	Push 3
5	39, 3, 4	Push 4
6	39, 0.75	Pop 3 and 4, compute 3 / 4 = 0.75, push 0.75
7	39.75	Pop 39 and 0.75, compute 39 + 0.75 = 39.75, push 39.75

2. What would be the complexity analysis for postfix evaluation?

Complexity Analysis:
Evaluating a postfix expression of length n using a stack takes **O(n)** time and **O(n)** space in the worst case. Each token (operand or operator) is processed once: operands are pushed, and operators pop two operands and push the result. Thus, the time and space complexity are both linear in the number of tokens.

3. Translate by hand the following infix expressions to postfix form: Show your working in the table below each expression.

a. 33 - 2 15 * 6
b. 11 * (6 + 2)
c. 17 + 3 - 5
d. 22 - 6 + 33 / 4

a. 33 - 2 15 * 6

Step	Stack (Operators)	Output (Postfix)	Explanation
1			Start
2		33	Output 33
3	-	33	Push '-'
4	-	33 2	Output 2
5	-	33 2 15	Output 15
6	- *	33 2 15	Push '*'
7	-	33 2 15 *	Pop '*' to output
8		33 2 15 * 6	Output 6
9		33 2 15 * 6 -	Pop '-' to output

Postfix: 33 2 15 * 6 - -

b. 11 * (6 + 2)

Step	Stack (Operators)	Output (Postfix)	Explanation
1			Start
2		11	Output 11
3	*	11	Push '*'
4	* (	11	Push '('
5	* (	11 6	Output 6
6	* (+	11 6	Push '+'
7	* (+	11 6 2	Output 2
8	*	11 6 2 +	Pop '+' to output, pop '('
9		11 6 2 + *	Pop '*' to output

Postfix: 11 6 2 + *

---

c. 17 + 3 - 5

Step	Stack (Operators)	Output (Postfix)	Explanation
1			Start
2		17	Output 17
3	+	17	Push '+'
4	+	17 3	Output 3
5	-	17 3 +	Pop '+', push '-'
6	-	17 3 + 5	Output 5
7		17 3 + 5 -	Pop '-' to output

Postfix: 17 3 + 5 -

---

d. 22 - 6 + 33 / 4

Step	Stack (Operators)	Output (Postfix)	Explanation
1			Start
2		22	Output 22
3	-	22	Push '-'
4	-	22 6	Output 6
5	+	22 6 -	Pop '-', push '+'
6	+	22 6 - 33	Output 33
7	+ /	22 6 - 33	Push '/'
8	+ /	22 6 - 33 4	Output 4
9	+	22 6 - 33 4 /	Pop '/' to output
10		22 6 - 33 4 / +	Pop '+' to output

Postfix: 22 6 - 33 4 / +

2. Perform a complexity analysis for a conversion of infix to postfix.

Complexity Analysis:
Converting an infix expression of length n to postfix using a stack takes **O(n)** time and **O(n)** space. Each token (operand, operator, or parenthesis) is processed once: operands are added to the output, operators and parentheses are pushed or popped from the stack. Thus, both time and space complexity are linear in the number of tokens.