Data Structure

ENCS205

School of Engineering & Technology (SOET)
K.R. MANGALAM University

Session 27: **Stacks :Quick Revision & Problem-Solving Boost**

Session 27 Outlook

- **Recap of Stacks**
- Quick recap of Push/Pop.-**Numerical Problem**
- Postfix Expression Evaluation **Numerical Problem**
- Infix to Postfix Conversion **Numerical Problem**
- Q&A
- Summarization

What is a Stack?

Definition: A Stack is a linear data structure that follows a particular order in which operations are performed. The order may be LIFO (Last In, First Out) or FILO (First In, Last Out).

Key Characteristics:

- Abstract Data Type (ADT): Defined by its behavior rather than its implementation.
- Restricted Access: Elements can only be added or removed from one end, called the TOP.

Stack Operations:

- Push
- Pop

Operations on Stack Numerical

Perform the following sequence of operations on an initially empty stack and show the stack's contents after each operation:

1. PUSH(10)
2. PUSH(20)
3. POP()
4. PUSH(30)
5. PUSH(40)
6. POP()
7. POP()

Operations on Stack Numerical

Solution:

Operation	Stack Contents (from bottom to top)
Initially Empty	[ ]
PUSH(10)	[10]
PUSH(20)	[10, 20]
POP()	[10] (20 is popped)
PUSH(30)	[10, 30]
PUSH(40)	[10, 30, 40]
POP()	[10, 30] (40 is popped)
POP()	[10] (30 is popped)

Postfix Expression Evaluation

Postfix: Operators appear after their operands (e.g., AB+).

Evaluation using Stack:

1. Scan expression left-to-right.
2. If operand, push onto stack.
3. If operator, pop two operands, perform operation, push result back.
4. Final result is the single value left on stack.

Postfix Expression Evaluation Numerical

Evaluate the postfix expression: 5 2 + 8 4 / - 3 *

Scanned Element	Operation/Stack State	Stack (bottom to top)
5	Push 5	[5]
2	Push 2	[5, 2]
+	Pop 2, Pop 5. Calculate 5 + 2 = 7. Push 7.	[7]
8	Push 8	[7, 8]
4	Push 4	[7, 8, 4]
/	Pop 4, Pop 8. Calculate 8 / 4 = 2. Push 2.	[7, 2]
-	Pop 2, Pop 7. Calculate 7 - 2 = 5. Push 5.	[5]
3	Push 3	[5, 3]
*	Pop 3, Pop 5. Calculate 5 * 3 = 15. Push 15.	[15]
Result	The final value on the stack is **15**.	[15]

Infix to Postfix Conversion

Infix: Standard notation with operators between operands (e.g., A + B).

Conversion using Stack: Uses an operator stack and considers operator precedence and associativity.

- **Operand:** Append to postfix string.
- **'(':** Push onto stack.
- **')':** Pop operators from stack and append to postfix until '(' is encountered (pop '(' too).
- **Operator:** Pop operators from stack and append to postfix while the stack's top operator has higher or equal precedence. Then push current operator.

Infix to Postfix Conversion

Precedence

(Higher to Lower):

1. ^ (Exponentiation)
2. *, /
3. +, -

Infix to Postfix Conversion Numerical

Convert the infix expression (A + B) * C - D / E to postfix.

Scanned Character	Operator Stack	Postfix Expression	Explanation
(	[( ]		Push '('
A	[( ]	A	Operand: Append to postfix.
+	[(, +]	A	Operator: Push '+'
B	[(, +]	A B	Operand: Append to postfix.
)	[ ]	A B +	Pop until '(': '+' popped.
*	[*]	A B +	Operator: Push '*'

Infix to Postfix Conversion Numerical

C - D / E

Scanned Character	Operator Stack	Postfix Expression	Explanation
C	[*]	A B + C	Operand: Append to postfix.
-	[-]	A B + C *	Operator '-': Pop '*' (higher precedence) then push '-'.
D	[-]	A B + C * D	Operand: Append to postfix.
/	[-, /]	A B + C * D	Operator '/': Push '/' (higher precedence than '-' on stack top).
E	[-, /]	A B + C * D E	Operand: Append to postfix.
End	[ ]	A B + C * D E / -	Pop remaining operators: '/' then '-'.

Final Postfix Expression: A B + C * D E / -

Stack (Arithmetic Conversion Practice Question)

Question 1: Translate the following infix expression to its postfix equivalent expression: A + B * (C + D) / F + D * E Also write the algorithm that you used for this conversion and evaluate the postfix expression's value for A=10, B=27, C=3 D=5 F=4 and ME= 2

Solutions :-

Algorithm (Converting Infix to Postfix)

1.**Initialize an empty stack** for operators and an empty list for the output (postfix expression).

2.**Read each token** (operand, operator, or parenthesis) from the infix expression from left to right:

   a. **If the token is an operand (A-Z, 0-9)**, append it to the output list.

   b. **If the token is an operator (+, -, *, /, etc.)**:

      a. While the stack is not empty and the top of the stack has an operator of greater than or equal precedence to the current token, pop operators from the stack to the output list.

      b. Push the current token onto the stack.

c. If the token is a left parenthesis '(', push it onto the stack.

d. If the token is a right parenthesis ')':

   a. Pop from the stack to the output list until a left parenthesis is encountered.
   b. Discard the left parenthesis from the stack.

3. After reading all tokens, pop any remaining operators from the stack to the output list.

1.The output list now contains the postfix expression.

Step	Read Token	Stack	Output (Postfix)
1	A		A
2	+	+	A
3	B	+	A B
4	*	+ *	A B
5	(	+ * (	A B
6	C	+ * (	A B C
7	+	+ * (+	A B C
8	D	+ * (+	A B C D
9	)	+ *	A B C D +
10	/	+ /	A B C D + *
11	F	+ /	A B C D + * F
12	+	+	A B C D + * F /
13	D	+	A B C D + * F / D
14	*	+ *	A B C D + * F / D
15	E	+ *	A B C D + * F / D E
16	End		A B C D + * F / + D E * +

Evaluating Postfix Expression Given values: A=10, B=27, C=3, D=5, F=4, E=2

Step	Read Token	Stack Before	Operation	Stack After
1	A (10)		Push 10	10
2	B (27)	10	Push 27	10 27
3	C (3)	10 27	Push 3	10 27 3
4	D (5)	10 27 3	Push 5	10 27 3 5
5	+	10 27 3 5	3 + 5 = 8	10 27 8
6	*	10 27 8	27 * 8 = 216	10 216
7	F (4)	10 216	Push 4	10 216 4
8	/	10 216 4	216 / 4 = 54	10 54
9	+	10 54	10 + 54 = 64	64
10	D (5)	64	Push 5	64 5
11	E (2)	64 5	Push 2	64 5 2
12	*	64 5 2	5 * 2 = 10	64 10
13	+	64 10	64 + 10 = 74	74

Arithmetic Conversion Practice Question

Question 2: What is the equivalent infix expression of the following postfix expression?

M, N, O, +, *, P, /, Q, R, S, T, /, +, *, –

Step 5 – Push Q, R, S, T onto the stack Stack – T, S, R, Q, ((M*(N+O))/P)

Step 6 – Pop T, S, R, Q and concatenate them with / and + and * Stack – ((Q*(R+(S/T))), ((M*(N+O))/P)

Step 7 – Pop the final expression from the stack after "-" Infix expression – (((M*(N+O))/P) – ((Q*(R+(S/T))))

Applications of Stacks

Stacks are fundamental in computer science for solving a wide range of problems.

Key Applications:

- **Expression Evaluation and Conversion:** Infix to Postfix/Prefix, Postfix Evaluation.
- **Recursion Management**: Function call stack.
- **Undo/Redo Functionality:** In text editors, software applications.
- **Backtracking Algorithms**: Solving mazes, knight's tour.
- **Browser History:** Navigating back through web pages.
- **Syntax Parsing:** Compilers use stacks to check for balanced parentheses, braces, etc.

Recursion

Definition: A programming technique where a function calls itself, directly or indirectly, to solve a problem.

How Stacks are Involved

The "Call Stack" (or "Execution Stack") is a crucial data structure used by the computer's operating system and programming language runtime to manage function calls.

- When a function is called, a **"stack frame"** (or "activation record") is pushed onto the call stack. This frame contains: **Local variables, Parameters & Return address**
- When the function finishes, its stack frame is popped from the call stack, and control returns to the address specified in the frame.

THANK
YOU

Assignment

1. Evaluate the following postfix expressions. Show each step of the evaluation, including the stack's state.

   a) 8 4 2 / + 5 *

   b) 10 2 3 * + 6 –

2. Convert the following infix expressions to their equivalent postfix form. Show the state of the operator stack and the postfix expression at each significant step. Assume the standard operator precedence:

   a) A + B * C – D

   b) (P - Q) * R + S / T