DATA STRUCTURE

Chapter 3 — STACKS

IMPORTANT PROBLEMS & SOLUTIONS

Frequently Asked Exam-Type Numerical & Theoretical Problems

(Diploma / B.Tech — Computer Science & Engineering)

How to Use This Problem Set

This booklet contains fully worked, step-by-step solutions to the types of problems on Stacks that are most frequently asked in semester examinations and competitive/government technical exams. Each problem is solved exactly the way it should be presented in an answer script — showing the stack status at every step, so that method marks as well as the final answer are covered. The problems are grouped under the following commonly examined categories:

- Infix to Postfix and Infix to Prefix conversion using stack
- Evaluation of Postfix and Prefix expressions
- Tracing PUSH / POP operations on array-based and linked-list-based stacks
- Checking balanced use of parentheses / brackets
- String reversal using a stack
- Recursion trace and activation record (stack frame) problems
- Tower of Hanoi — recursion and stack-based reasoning

Attempt each problem on your own first, then check the given solution and match your working step by step.

Problem 1

Statement: Convert the following infix expression to postfix notation using a stack, showing the stack contents at every step: A + B * C - (D / E ^ F)

Solution:

Precedence (highest to lowest): ^ (right associative) > *, / > +, -. Scan the expression from left to right, using a stack to hold operators.

Token	Action	Stack (top → right)	Output so far
A	Output	(empty)	A
+	Push	+	A
B	Output	+	A B
*	Push (higher prec. than +)	+ *	A B
C	Output	+ *	A B C
-	Pop * , pop + ; push -	-	A B C * +
(	Push	- (	A B C * +
D	Output	- (	A B C * + D
/	Push	- (/	A B C * + D
E	Output	- (/	A B C * + D E
^	Push (higher prec. than /)	- (/ ^	A B C * + D E
F	Output	- (/ ^	A B C * + D E F
)	Pop ^, / till '(' ; discard '('	-	A B C * + D E F ^ /
End	Pop remaining -	(empty)	A B C * + D E F ^ / -

Answer: A B C * + D E F ^ / -

Problem 2

Statement: Convert the following infix expression to prefix notation: (A - B) / (C + D * E)

Solution:

Method: (i) Reverse the infix expression and interchange every '(' with ')' and every ')' with '('. (ii) Convert the resulting expression to postfix using the normal algorithm. (iii) Reverse the postfix result to obtain the prefix expression.

Step 1 — Reverse and swap brackets:

```
Original :  ( A - B ) / ( C + D * E )
Modified :  ( E * D + C ) / ( B - A )
```

Step 2 — Convert the modified expression to postfix:

Token	Action	Stack (top → right)	Output so far
(	Push	(	
E	Output	(	E
*	Push	(*	E
D	Output	(*	E D
+	Pop * (higher prec.); push +	(+	E D *
C	Output	(+	E D * C
)	Pop + ; discard '('	(empty)	E D * C +
/	Push	/	E D * C +
(	Push	/ (	E D * C +
B	Output	/ (	E D * C + B
-	Push	/ (-	E D * C + B
A	Output	/ (-	E D * C + B A
)	Pop - ; discard '('	/	E D * C + B A -
End	Pop remaining /	(empty)	E D * C + B A - /

Step 3 — Reverse the postfix result:

```
Postfix of modified expression :  E D * C + B A - /
Reversed (= Prefix)            :  / - A B + C * D E
```

Answer: / - A B + C * D E

Problem 3

Statement: Evaluate the postfix expression: 6 5 2 3 + 8 * + 3 + * using a stack.

Solution:

Rule: scan left to right; push every operand; on encountering an operator, pop the required number of operands, apply the operator, and push the result back.

Token	Action	Stack (bottom → top)
6	Push 6	6
5	Push 5	6, 5
2	Push 2	6, 5, 2

3	Push 3	6, 5, 2, 3
+	Pop 3, 2 → 2+3=5 → Push 5	6, 5, 5
8	Push 8	6, 5, 5, 8
*	Pop 8, 5 → 5*8=40 → Push 40	6, 5, 40
+	Pop 40, 5 → 5+40=45 → Push 45	6, 45
3	Push 3	6, 45, 3
+	Pop 3, 45 → 45+3=48 → Push 48	6, 48
*	Pop 48, 6 → 6*48=288 → Push 288	288

Answer: 288

Problem 4

Statement: Evaluate the prefix expression: + * 2 3 - 4 1 using a stack.

Solution:

Rule for prefix evaluation: scan the expression from right to left; push every operand; on encountering an operator, pop the first operand as op1 and the second as op2, compute (op1 operator op2), and push the result back.

Token (R→L)	Action	Stack (bottom → top)
1	Push 1	1
4	Push 4	1, 4
-	Pop op1=4, op2=1 → 4-1=3 → Push 3	3
3	Push 3	3, 3
2	Push 2	3, 3, 2
*	Pop op1=2, op2=3 → 2*3=6 → Push 6	3, 6
+	Pop op1=6, op2=3 → 6+3=9 → Push 9	9

Verification: the expression represents (2 * 3) + (4 - 1) = 6 + 3 = 9, which matches the stack-based evaluation.

Answer: 9

Problem 5

Statement: A stack is implemented using an array of size 5 (MAXSIZE = 5, valid indices 0 to 4). Trace the contents of the stack and the value of TOP for the following sequence of operations: PUSH(10), PUSH(20), PUSH(30), POP(), PUSH(40), PUSH(50), PUSH(60), PUSH(70), POP(), POP().

Solution:

Operation	Result	Stack Contents	TOP
PUSH(10)	Inserted	10	0
PUSH(20)	Inserted	10, 20	1
PUSH(30)	Inserted	10, 20, 30	2
POP()	Returns 30	10, 20	1
PUSH(40)	Inserted	10, 20, 40	2

PUSH(50)	Inserted	10, 20, 40, 50	3
PUSH(60)	Inserted	10, 20, 40, 50, 60	4 (stack full)
PUSH(70)	STACK OVERFLOW — TOP = MAXSIZE-1, so 70 is rejected	10, 20, 40, 50, 60	4
POP()	Returns 60	10, 20, 40, 50	3
POP()	Returns 50	10, 20, 40	2

Answer: Final stack (bottom to top): 10, 20, 40 | TOP = 2 | one overflow occurred while pushing 70.

Problem 6

Statement: Check whether the following expression has correctly balanced and nested symbols, using a stack: { (A + B) * [C - D] }

Solution:

Rule: scan left to right; push every opening symbol; on every closing symbol, pop the stack and check that it matches the corresponding opening symbol. The expression is balanced only if the stack becomes empty exactly at the end and no mismatch occurs.

Symbol Scanned	Action	Stack (bottom → top)
{	Push	{
(	Push	{, (
)	Pop '(' — matches	{
[	Push	{, [
]	Pop '[' — matches	{
}	Pop '{' — matches	(empty)

Answer: Stack is empty at the end with no mismatch — the expression is BALANCED.

Problem 7

Statement: Check whether the following expression is balanced, using a stack: (A + [B * C)]

Solution:

Symbol Scanned	Action	Stack (bottom → top)
(	Push	(
[	Push	(, [
)	Pop top → '[' , but ')' should match '(' → MISMATCH	(stopped — error)

Answer: Mismatch detected — the expression is NOT balanced (the closing ')' does not match the innermost opening '[').

Problem 8

Statement: A stack is implemented using a singly linked list, with TOP pointing to the first node. Trace the list and the value returned after each of the following operations: PUSH(A), PUSH(B), PUSH(C), POP(), PUSH(D), POP(), POP().

Solution:

Operation	Value Returned	Linked List (TOP → ... → NULL)
PUSH(A)	—	A → NULL
PUSH(B)	—	B → A → NULL
PUSH(C)	—	C → B → A → NULL
POP()	C	B → A → NULL
PUSH(D)	—	D → B → A → NULL
POP()	D	B → A → NULL
POP()	B	A → NULL

Answer: Final list: A → NULL (TOP points to A). Values returned by the three POP() calls, in order: C, D, B.

Problem 9

Statement: Write the logic to reverse a string using a stack, and trace it for the string "STACK".

Solution:

```
Algorithm REVERSE_STRING(STR)
Step 1: For each character CH in STR, from left to right:
           PUSH(CH)
Step 2: Set RESULT = empty string
Step 3: While stack is NOT empty:
           CH = POP()
           Append CH to RESULT
Step 4: Return RESULT
```

Trace for STR = "STACK":

Step	Stack after pushing (bottom → top)	Character Popped	Result so far
Push S, T, A, C, K	S, T, A, C, K	—	
Pop 1	S, T, A, C	K	K
Pop 2	S, T, A	C	KC
Pop 3	S, T	A	KCA
Pop 4	S	T	KCAT
Pop 5	(empty)	S	KCATS

Answer: KCATS

Problem 10

Statement: Write the recursive algorithm for the Tower of Hanoi problem and list the complete sequence of moves for 3 disks, moving them from peg A to peg C using peg B as auxiliary. Explain how the recursive calls are managed using a stack.

Solution:

```
Algorithm TOH(n, SOURCE, AUX, DEST)
Step 1: If n = 1, then
           Move disk 1 from SOURCE to DEST
           Return
Step 2: TOH(n - 1, SOURCE, DEST, AUX)
Step 3: Move disk n from SOURCE to DEST
Step 4: TOH(n - 1, AUX, SOURCE, DEST)
```

Each call to TOH pushes a new activation record (storing n, SOURCE, AUX, DEST and the return address) onto the call stack; the record is popped only after both of its recursive sub-calls have completed. For n = 3, calling TOH(3, A, B, C) generates the following sequence of disk moves, in order:

1. Move disk 1 from A to C
2. Move disk 2 from A to B
3. Move disk 1 from C to B
4. Move disk 3 from A to C
5. Move disk 1 from B to A
6. Move disk 2 from B to C
7. Move disk 1 from A to C

The total number of moves generated by the recursion is always $2^n - 1$; for n = 3 this gives $2^3 - 1 = 7$ moves, which matches the sequence above.

Answer: 7 moves in total, as listed above; minimum possible moves for 3 disks = $2^3 - 1 = 7$.

Problem 11

Statement: Using the recursive definition fact(n) = n * fact(n - 1), with fact(0) = 1, trace the activation records placed on the stack for the call fact(5), and show how the final result is obtained.

Solution:

Calling phase — an activation record is pushed for every call, from fact(5) down to the base case fact(0):

Call (pushed, top → bottom order at deepest point)	Waiting to Compute
fact(0) (TOP — base case)	returns 1 directly
fact(1)	1 * fact(0)
fact(2)	2 * fact(1)
fact(3)	3 * fact(2)
fact(4)	4 * fact(3)
fact(5) (bottom — first call made)	5 * fact(4)

Returning phase — activation records are popped one at a time, from the top (fact(0)) downward, each returning its value to the call that is waiting for it:

8. fact(0) returns 1 → popped
9. fact(1) = 1 * 1 = 1 → popped
10. fact(2) = 2 * 1 = 2 → popped
11. fact(3) = 3 * 2 = 6 → popped
12. fact(4) = 4 * 6 = 24 → popped
13. fact(5) = 5 * 24 = 120 → popped (call stack now empty)

Answer: fact(5) = 120

Practice Tip: In the examination, always draw the stack-status table (as shown above) for full step marks — do not write only the final answer.