DATA STRUCTURE

Chapter 3 — STACKS

IMPORTANT PROBLEMS & SOLUTIONS

Set 4 (Problems 31–39)

Prefix-to-Infix, Radix Conversion, Span & Nearest-Element Problems, Trees, Recursion and Stack-Queue Interconversion

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

About This Set

This fourth set completes the circle of expression conversions and introduces several well-known stack-based problems that frequently appear as long or practical questions — nearest-element problems, the stock span problem, palindrome checking, expression trees, and interconversion between stacks and queues. Problem numbering continues from Sets 1–3. This set covers:

- Prefix to Infix conversion (completing the full circle of expression conversions)
- Decimal to Octal and Decimal to Hexadecimal conversion using a stack
- Next Smaller Element — the mirror problem to Next Greater Element
- The Stock Span Problem, solved efficiently with a stack
- Checking whether a string is a palindrome using a stack
- Building an expression tree from a postfix expression using a stack
- Deleting the middle element of a stack using only recursion
- Simulating browser Back / Forward navigation with two stacks
- Implementing a Stack using a single Queue (the reverse of Set 3's Queue-from-two-Stacks problem)

Problem 31

Statement: Convert the prefix expression - * A + B C D to infix notation, using a stack.

Solution:

Rule: scan the expression from right to left; push every operand. On encountering an operator, pop the first element as op1 and the second as op2, form the string "(op1 operator op2)", and push it back as a single unit.

Token (R→L)	Action	Stack (bottom → top)
D	Push	D
C	Push	D, C
B	Push	D, C, B
+	Pop op1=B, op2=C → form (B+C) → Push	D, (B+C)
A	Push	D, (B+C), A
*	Pop op1=A, op2=(B+C) → form (A*(B+C)) → Push	D, (A*(B+C))
-	Pop op1=(A*(B+C)), op2=D → form ((A*(B+C))-D) → Push	((A*(B+C))-D)

Notice that this is exactly the reverse of Problem 14 in Set 2, which converted the postfix form of this same expression back into prefix — a good way to cross-check both conversions.

Answer: ((A*(B+C))-D)

Problem 32

Statement: Convert the decimal number 478 to (a) octal and (b) hexadecimal, using a stack.

Solution:

The same divide-and-stack method used for binary conversion (Problem 22) works for any base: repeatedly divide by the target base, push the remainder, and continue until the number becomes 0. Reading the stack from top to bottom then gives the digits in the correct order.

(a) Converting 478 to Octal (base 8):

N (before)	N mod 8 (pushed)	N div 8 (new N)
478	6	59
59	3	7
7	7	0

Popping the stack gives digits 7, 3, 6. Check: 7×64 + 3×8 + 6 = 448 + 24 + 6 = 478 ✓

Answer: 478 (decimal) = 736 (octal)

(b) Converting 478 to Hexadecimal (base 16):

N (before)	N mod 16 (pushed)	N div 16 (new N)
478	14 (=E)	29
29	13 (=D)	1
1	1	0

Popping the stack gives digits 1, D, E. Check: 1×256 + 13×16 + 14 = 256 + 208 + 14 = 478 ✓

Answer: 478 (decimal) = 1DE (hexadecimal)

Problem 33

Statement: For the array [4, 5, 2, 10, 8], find the Next Smaller Element (NSE) for every element — the first element to its right that is smaller than it (or -1 if none exists) — using a stack.

Solution:

Rule: scan the array from right to left. For each element, pop the stack while its top is greater than or equal to the current element. If the stack becomes empty, the NSE is -1; otherwise, the NSE is the new stack top. Then push the current element.

Element (R→L)	Popped (≥ current)	NSE Found	Stack after step (bottom → top)
8	none (stack empty)	-1	8
10	none (8 < 10)	8	8, 10
2	10, 8 (both ≥ 2)	-1	2
5	none (2 < 5)	2	2, 5
4	5 (5 ≥ 4)	2	2, 4

Reading the NSE values in the original left-to-right order of the array [4, 5, 2, 10, 8]:

Answer: NSE array: [2, 2, -1, 8, -1]

Problem 34

Statement: For the stock prices [100, 80, 60, 70, 60, 75, 85] (day 0 to day 6), compute the span of each day using a stack. The span of a day is the number of consecutive days up to and including that day for which the price was less than or equal to that day's price.

Solution:

Maintain a stack of day-indices. For each day i, pop all indices whose price is less than or equal to price[i] (they can never block the span). The span is i minus the index now on top of the stack (or i+1 if the stack becomes empty); then push index i.

Day (Price)	Indices Popped	Stack Top After Pop	Span
0 (100)	— (stack was empty)	—	1
1 (80)	none (100 > 80)	day 0 (100)	1 - 0 = 1
2 (60)	none (80 > 60)	day 1 (80)	2 - 1 = 1
3 (70)	day 2 (60 ≤ 70)	day 1 (80)	3 - 1 = 2
4 (60)	none (70 > 60)	day 3 (70)	4 - 3 = 1
5 (75)	day 4 (60), day 3 (70)	day 1 (80)	5 - 1 = 4
6 (85)	day 5 (75), day 1 (80)	day 0 (100)	6 - 0 = 6

Note on day 6: popping stops at day 0 (price 100) because 100 is greater than 85, so day 0 is never popped — the span is 6 − 0 = 6.

Answer: Span array (day 0 to day 6): [1, 1, 1, 2, 1, 4, 6]

Problem 35

Statement: Check whether the string "MALAYALAM" is a palindrome, using a stack.

Solution:

Rule: push the characters of the first half of the string onto a stack. If the length is odd, skip the middle character. Then compare the second half of the string, character by character, with the characters popped from the stack — if every comparison matches, the string is a palindrome.

MALAYALAM has 9 characters (indices 0–8): M A L A Y A L A M. The first half (indices 0–3) is pushed: M, A, L, A. The middle character (index 4, 'Y') is skipped.

Second-Half Character (index)	Popped from Stack	Match?
A (index 5)	A	Yes
L (index 6)	L	Yes
A (index 7)	A	Yes
M (index 8)	M	Yes

Answer: All characters match — "MALAYALAM" IS a palindrome.

Problem 36

Statement: Construct the expression tree for the postfix expression A B + C D - * using a stack, and draw the resulting tree.

Solution:

Rule: scan left to right; push a leaf node for every operand. On encountering an operator, pop two nodes from the stack — the first popped becomes the right child and the second popped becomes the left child of a new node labelled with the operator — then push this new node back onto the stack. At the end, the single node left on the stack is the root of the expression tree.

Token	Action	Stack (bottom → top)
A	Push leaf node A	A
B	Push leaf node B	A, B

+	Pop B (right), A (left) → new node '+' → Push	(+)
C	Push leaf node C	(+), C
D	Push leaf node D	(+), C, D
-	Pop D (right), C (left) → new node '-' → Push	(+), (-)
*	Pop (-) (right), (+) (left) → new node '*' → Push	(*) ← root

The resulting expression tree:

```
            *
          /   \
        +       -
       / \     / \
      A   B   C   D
```

Answer: Root = *, left subtree = (+ with children A, B), right subtree = (- with children C, D).

Problem 37

Statement: Delete the middle element of the stack S (top → bottom) = 1, 2, 3, 4, 5 using only recursion (no auxiliary stack or array).

Solution:

For a stack of n elements, the middle element (counted from the top, 1-indexed) is at position (n/2) + 1 using integer division. For n = 5, the target position is (5/2) + 1 = 2 + 1 = 3 — i.e. the 3rd element from the top, which is the value 3 in this stack.

```
Algorithm DELETE_MID(S, curr, target)
Step 1: If curr = target, then
          Pop(S) and discard it   // middle element removed
          Return
Step 2: temp = Pop(S)
Step 3: DELETE_MID(S, curr + 1, target)
Step 4: Push(temp) back onto S

Initial call: DELETE_MID(S, 1, 3)   // for n = 5, target = (5/2)+1 = 3
```

Trace for S (top → bottom) = 1, 2, 3, 4, 5:

1. curr=1 ≠ 3: pop 1 (temp=1); S becomes 2, 3, 4, 5; recurse with curr=2.
2. curr=2 ≠ 3: pop 2 (temp=2); S becomes 3, 4, 5; recurse with curr=3.
3. curr=3 = 3: pop and discard 3 — this is the middle element; S becomes 4, 5; return.
4. Unwinding: push temp=2 back → S (top → bottom) = 2, 4, 5.
5. Unwinding: push temp=1 back → S (top → bottom) = 1, 2, 4, 5.

Answer: Final stack (top → bottom): 1, 2, 4, 5 — the middle element 3 has been removed.

Problem 38

Statement: Using a BackStack and a ForwardStack, trace the current page shown after the sequence: start at page A; Visit(B); Visit(C); Visit(D); Back(); Back(); Forward(); Visit(E).

Solution:

Rule: Visit(new page) pushes the current page onto BackStack and makes the new page current (and clears ForwardStack, since visiting a new page discards any forward history). Back() pushes the current page onto

ForwardStack and makes the top of BackStack (popped) the new current page. Forward() pushes the current page onto BackStack and makes the top of ForwardStack (popped) the new current page.

Action	New Current Page	BackStack (bottom → top)	ForwardStack (bottom → top)
Start	A	(empty)	(empty)
Visit(B)	B	A	(empty)
Visit(C)	C	A, B	(empty)
Visit(D)	D	A, B, C	(empty)
Back()	C	A, B	D
Back()	B	A	D, C
Forward()	C	A, B	D
Visit(E)	E	A, B, C	(empty)

Answer: Final current page: E. BackStack: A, B, C. ForwardStack: empty (cleared by the new visit).

Problem 39

Statement: Explain how a stack can be implemented using only one queue, Q, and trace: Push(1), Push(2), Push(3), Pop(), Push(4), Pop().

Solution:

```
PUSH(ITEM):
   Enqueue(ITEM) onto Q
   Repeat (Size(Q) - 1) times:
        Dequeue an element from Q and Enqueue it again at the back
   // this rotation brings the newly pushed item to the front of Q

POP():
   Return Dequeue(Q)   // the front of Q is always the most recently pushed item
```

Every PUSH costs extra rotation work so that POP stays a simple, fast dequeue from the front — the opposite trade-off from Problem 24's queue-using-two-stacks, where PUSH (enqueue) was cheap and POP occasionally did the expensive work.

Operation	Result	Queue Q (front → back)
Push(1)	—	1
Push(2)	—	2, 1
Push(3)	—	3, 2, 1
Pop()	Returns 3	2, 1
Push(4)	—	4, 2, 1
Pop()	Returns 4	2, 1

Answer: Pop() calls return 3, then 4 — correct LIFO order for the pushes 1, 2, 3, then 4.

Practice Tip: For nearest-element and span problems, always be clear about the scan direction (left-to-right or right-to-left) and the exact pop condition (>, <, ≥, or ≤) — a single wrong comparison symbol changes the entire answer.