FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XII · CODE 308

Stack

CUET unit: Stack

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- A Stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle, with its own definition, real-life analogies, operations, and Python implementation.
- Two fundamental operations — PUSH (insert) and POP (delete) — and their exception conditions (overflow and underflow) are explained with step-by-step diagrams.
- Python's built-in list with `append()` and `pop()` methods is used to implement a stack without needing an explicit TOP pointer.
- Stacks apply to arithmetic expression notations (Infix, Prefix/Polish, Postfix/Reverse Polish) and drive Infix-to-Postfix conversion (Algorithm 3.1) and Postfix evaluation (Algorithm 3.2).
- CUET tests stacks heavily through code-output questions on PUSH/POP sequences, conversion of expressions, and identification of overflow/underflow conditions.

📖 Detailed Notes

2.1 Core concepts

- **Data structure definition:** A data structure defines a mechanism to store, organise, and access data along with operations that can be efficiently performed on it. String, List, Set, Tuple are sequence data types; Stack and Queue are two other popular data structures used in programming. (NCERT §3.1, p. 39)
- **Stack definition:** A stack is a linear data structure in which elements are added and removed from the same end called the TOP. It follows the LIFO (Last-In-First-Out) principle — the element inserted last is the first one to be removed. Real-life analogies: pile of plates, stack of books, pile of clothes in an almirah, bangles worn on a wrist. (NCERT §3.2, p. 40)
- **Linear data structure:** A data structure in which elements are organised in a sequence is called a linear data structure. Other examples include Array, Linked List, Queue. (NCERT §3.2 sidebar, p. 40)
- **Applications of stack in programming:** (a) Reversing a string — characters are pushed onto the stack and popped in reverse order; (b) Undo/redo in text/image editors — the system uses a stack to keep track of changes; (c) Browser BACK

button — history of visited pages is maintained as a stack; (d) Parentheses matching — compiler uses a stack to verify that each opening parenthesis has a corresponding closing parenthesis. (NCERT §3.2.1, p. 41)

- **PUSH operation:** PUSH adds a new element at the TOP of the stack (insertion operation). Adding an element to a full stack results in an exception called **overflow**. (NCERT §3.3.1, p. 42)
- **POP operation:** POP removes the topmost element from the stack (deletion operation). Trying to delete an element from an empty stack results in an exception called **underflow**. (NCERT §3.3.1, p. 42)
- **Figure 3.2:** Illustrates ten sequential states of a stack — Empty Stack, Push 1, Push 2, Pop (2 removed), Push 3, Push 4, Pop (4 removed), Pop (3 removed), Pop (1 removed), Empty Stack — demonstrating LIFO behaviour. (NCERT §3.3.1, p. 42)
- **Implementation of Stack in Python:** Python's `list` data type is used to implement a stack. The built-in methods `append()` (for PUSH) and `pop()` (for POP) operate at the rightmost end of the list, so no explicit TOP variable is needed. An implemented Python list-based stack will never face overflow unless the system runs out of memory. (NCERT §3.4, p. 43)
- **Python stack functions:** `isEmpty(glassStack)` — returns True if stack is empty; `opPush(glassStack, element)` — uses `append()` to push; `size(glassStack)` — uses `len()` to return count; `top(glassStack)` — returns `glassStack[len-1]` without removing; `opPop(glassStack)` — uses `pop()` to remove and return top element, prints 'underflow' if empty; `display(glassStack)` — prints elements from top to bottom using `range(x-1, -1, -1)`. (NCERT §3.4, pp. 43–45)
- **Three notations for arithmetic expressions:** Any arithmetic expression can be written in three forms — **Infix** (operator between operands, e.g., x * y + z), **Prefix/ Polish** (operator before operands, introduced by Jan Lukasiewicz in the 1920s, e.g., +zxy), and Postfix/Reverse Polish (operator after operands, e.g., xyz+). Prefix and Postfix expressions do not require parentheses because operator order is determined by position. (NCERT §3.5, p. 46–47)
- **Table 3.1 — Infix, Prefix, Postfix:** Summarises the three notations with examples such as Infix `(x+y)/(z*5)`, Prefix `/+xy*z5`, Postfix `xy+z5*/`. (NCERT §3.5, p. 47)
- **Infix-to-Postfix conversion using a stack (Algorithm 3.1):** A stack tracks operators; a string variable `postExp` accumulates the output. Rules: operands are appended directly; left parenthesis is pushed; right parenthesis causes popping until left parenthesis is found (both discarded); operator is pushed if its precedence is greater than or equal to top of stack, otherwise pop operators of higher/equal precedence first; at end, pop all remaining operators. (NCERT §3.6, pp. 47–48)
- **Example 3.1:** Conversion of `(x+y)/(z*8)` to postfix `xy+z8*/` is traced step by step using Figure 3.3. (NCERT §3.6, p. 48–49)
- **Evaluation of Postfix expression using a stack (Algorithm 3.2):** Operands are pushed onto the stack; when an operator is encountered, two operands are popped,

the operator is applied, and the result is pushed back. At end, if the stack has a single element, that is the result; otherwise the postfix expression is invalid. (NCERT §3.7, p. 49)

- **Example 3.2:** Evaluation of postfix expression `7 8 2 * 4 / +` gives result 11, traced through Figure 3.4. Key steps: Push 7, Push 8, Push 2, encounter `*` → pop 8 and 2, push 16; encounter `4` → Push 4; encounter `/` → pop 16 and 4, push 4; encounter `+` → pop 7 and 4, push 11. Final stack has single element 11. (NCERT §3.7, p. 50)
- **Conversion vs Evaluation — what is PUSHed:** During Infix-to-Postfix conversion only **operators** are pushed onto the stack; during Postfix evaluation only **operands** are pushed onto the stack. (NCERT §3.7 summary, p. 51)
- **Why stacks underpin function calls.** Though beyond the NCERT syllabus here, modern programming languages implement function-call return addresses on a call stack — the same LIFO discipline. Understanding `push` / `pop` deepens intuition for recursion (CUET Class XII context).
- **Why prefix/postfix avoid parentheses (NCERT §3.5, p. 47).** In infix `a + b * c` needs parentheses to force `(a + b) * c`. In postfix the same intent is `a b + c *` and the alternative is `a b c * +` — order alone unambiguously decides evaluation, so parentheses are never required.
- **Time complexity of stack ops (NCERT §3.4).** Both `push` (`append`) and `pop` are O(1) when operating at the rightmost end of a Python list — that is why Python lists are ideal for stack implementation. Inserting at the front would be O(n).

2.2 Definitions to memorise

Term	Definition	Page
Stack	A linear data structure where insertion and deletion are done from one end only (TOP), following LIFO order	40
LIFO	Last-In-First-Out; the element inserted last is the first to be removed	40
TOP	The end of a stack from which elements are added or removed	42
PUSH	Insertion operation that adds a new element at the TOP of the stack	42
POP	Deletion operation that removes the topmost element from the stack	42
Overflow	Exception raised when trying to PUSH an element onto a full stack	42
Underflow	Exception raised when trying to POP an element from an empty stack	42
Infix notation		46

Term	Definition	Page
	Arithmetic expression format where operator is placed between the operands (e.g., x + y)	
Prefix (Polish) notation	Arithmetic expression format where operator is placed before its operands (e.g., +xy)	46
Postfix (Reverse Polish) notation	Arithmetic expression format where operator is placed after its operands (e.g., xy+)	46–47
Linear data structure	A data structure in which elements are organised in a sequence	40
Data structure	A mechanism to store, organise, and access data along with efficient operations on that data	39
`isEmpty()`	Stack helper returning True when the stack has no elements	44
`top()`	Stack helper returning the topmost element without removing it (also called peek)	44
`size()`	Stack helper returning the number of elements currently in the stack	44
`display()`	Stack helper that prints elements from top to bottom	45
Polish notation	Another name for Prefix notation (operator before operands)	46
Reverse Polish notation	Another name for Postfix notation (operator after operands)	46-47
BODMAS	Conventional precedence rule needed for infix evaluation; postfix/prefix do not need it	47
Operator precedence	Order in which arithmetic operators are evaluated; controls how a stack pops during conversion	47-48
Algorithm 3.1	NCERT's stack-based algorithm for Infix-to-Postfix conversion	47-48
Algorithm 3.2	NCERT's stack-based algorithm for Postfix expression evaluation	49
Linked List	A linear data structure; mentioned in §3.2 sidebar alongside arrays and queues	40
Queue	A linear data structure following FIFO; companion concept introduced in §3.1	39

2.3 Diagrams / processes to remember

- **Figure 3.1 (p. 40):** Stack of plates and stack of books illustrating the real-life analogy for a stack — elements added/removed only from the top.
- **Figure 3.2 (p. 42):** Ten-state diagram of PUSH and POP operations on a stack of glasses numbered 1–4. Shows Empty Stack → Push 1 → Push 2 → Pop (2 removed) → Push 3 → Push 4 → Pop (4 removed) → Pop (3 removed) → Pop (1 removed) → Empty Stack. Essential for tracing LIFO order in MCQs.

- **Figure 3.3 (p. 48–49):** Step-by-step conversion of infix expression `(x+y)/(z*8)` to postfix `xy+z8*/` using Algorithm 3.1. Shows the stack state and postExp string after processing each symbol.
- **Figure 3.4 (p. 50):** Step-by-step evaluation of postfix expression `7 8 2 * 4 / +` using Algorithm 3.2. Shows stack state after each symbol; final result = 11.
- **Table 3.1 (p. 47):** Three-row table summarising Infix, Prefix, and Postfix notations with descriptions and examples — must be memorised for notation-identification questions.

2.4 Common confusions / NTA trap points

- **Overflow vs Underflow:** Students frequently swap these. Overflow happens on PUSH to a FULL stack; Underflow happens on POP from an EMPTY stack. In Python list implementation, overflow practically never occurs (unlimited memory), but underflow still can.
- **PUSH in conversion vs evaluation:** During Infix-to-Postfix conversion, only OPERATORS are pushed onto the stack (operands go directly to output). During Postfix evaluation, only OPERANDS are pushed. NTA often asks "what is pushed during evaluation/conversion" — these are opposite behaviours.
- **append() and pop() as PUSH and POP:** In Python, `list.append(x)` implements PUSH and `list.pop()` implements POP (removes from the rightmost end). Students confuse `pop()` (removes last element) with deletion from left.
- **Prefix introduced by Jan Lukasiewicz:** NTA may ask who introduced Polish/Prefix notation. Answer: Polish mathematician Jan Lukasiewicz in the 1920s. Do not confuse with Postfix (Reverse Polish), which reverses his logic.
- **Single traversal sufficiency (NCERT §3.5, p. 47).** Prefix and Postfix expressions can be evaluated in a single left-to-right traversal because operator precedence is encoded in position — no parentheses needed.
- **TOP element is the LAST pushed (NCERT §3.3.1, p. 42).** It is the most recent insertion. NTA distractor: claims top is the oldest element.
- **Order of operands during evaluation (NCERT Algorithm 3.2, p. 49).** When `*` is encountered after pushing 8 then 2: the FIRST popped (2) is the right operand, the SECOND popped (8) is the left operand → 8*2=16. Swapping order leads to subtraction/division errors.
- **Postfix never needs parentheses (NCERT §3.5, p. 47).** Adding parentheses to a postfix expression is meaningless.
- **Stack stores OPERATORS during conversion (NCERT §3.6, p. 47).** Operands flow straight to output; only operators go through the stack.
- **In Python, `list.pop()` defaults to last index (NCERT §3.4, p. 43).** `pop()` with no arg = pop last (correct for stack).

- **A queue is NOT a stack (NCERT §3.1 sidebar, p. 39).** Queue is FIFO; Stack is LIFO. NTA may swap these.

🎯 Practice MCQs

Q1. Which of the following correctly describes the LIFO principle of a stack?

A. The element inserted first will be the first one to be removed.

B. The element inserted last will be the last one to be removed.

C. The element inserted last will be the first one to be removed.

D. Elements can be removed from both ends of the structure.

Q2. In Python, a stack is implemented using a list. Which pair of built-in list methods is used for the PUSH and POP operations respectively?

A. `insert()` and `remove()`

B. `append()` and `pop()`

C. `add()` and `delete()`

D. `push()` and `pull()`

Q3. Consider the following statements about stack operations: **Statement I:** Trying to PUSH an element onto a full stack raises an exception called underflow. **Statement II:** Trying to POP an element from an empty stack raises an exception called overflow. Which of the above statements is/are correct?

A. Only Statement I

B. Only Statement II

C. Both Statement I and Statement II

D. Neither Statement I nor Statement II

PYQ Alignment

Stacks have a consistent presence in CUET Computer Science papers, with questions typically testing PUSH/POP trace outputs (code-output format), identification of overflow/underflow conditions, conversion between Infix and Postfix notations, and matching operations to their correct Python list methods. Expression evaluation questions using numeric postfix expressions (similar to Example 3.2) appear frequently as medium-difficulty items. See PYQ archive for Computer Science.