Polish Expression Using Stack

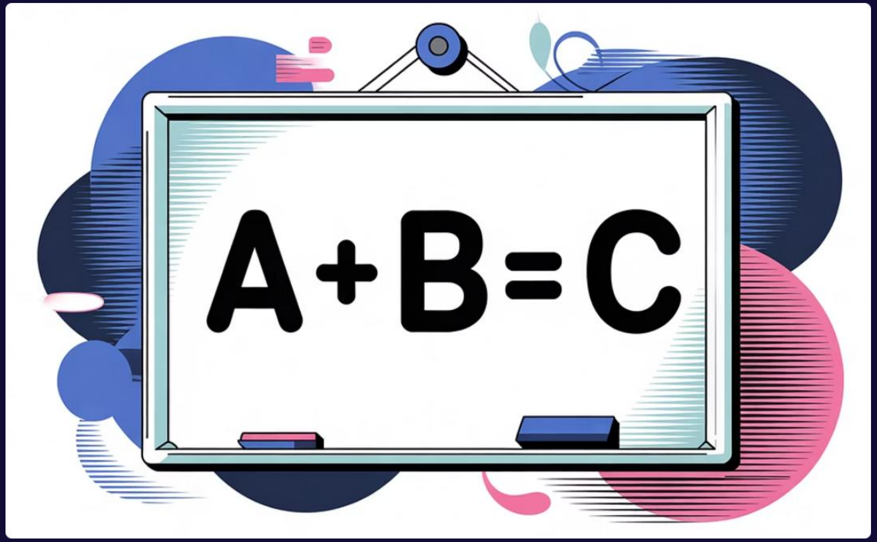
Course: Computer Engineering (CE–B) — Presented by Chanpura Keyur D., Chanpura Shivam M., Maheriya Hardik S., Wadhvana Shyam B.

Why study expression conversion?

Expressions encode arithmetic and logical computations. Converting between notations and evaluating them efficiently is foundational for compilers, interpreters, and calculators.

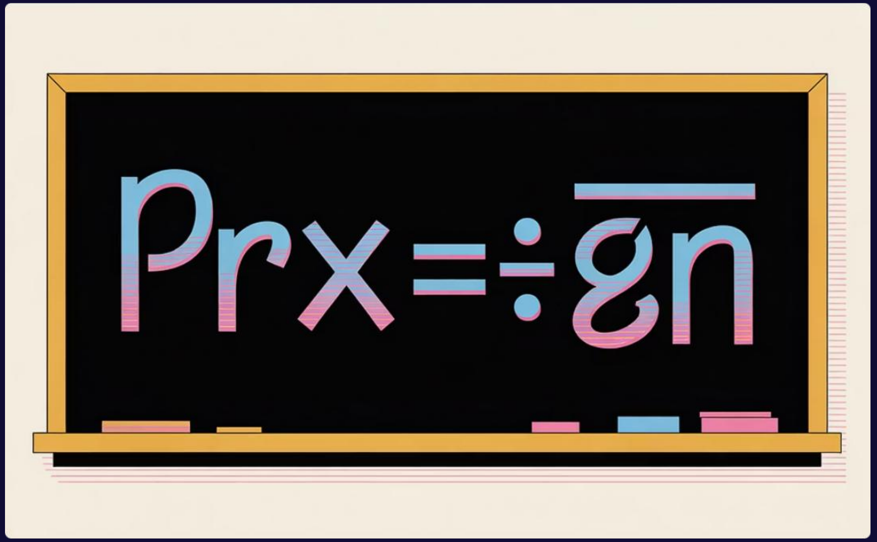
- Enables stack-based evaluation on limited-memory devices
- Removes ambiguity from parentheses and grouping

Expression Notations — Quick overview



Infix

Operator between operands. Human-friendly but requires precedence rules and parentheses for disambiguation.



Prefix (Polish)

Operator appears before its operands. No parentheses needed when arity is known.

Postfix (Reverse Polish)

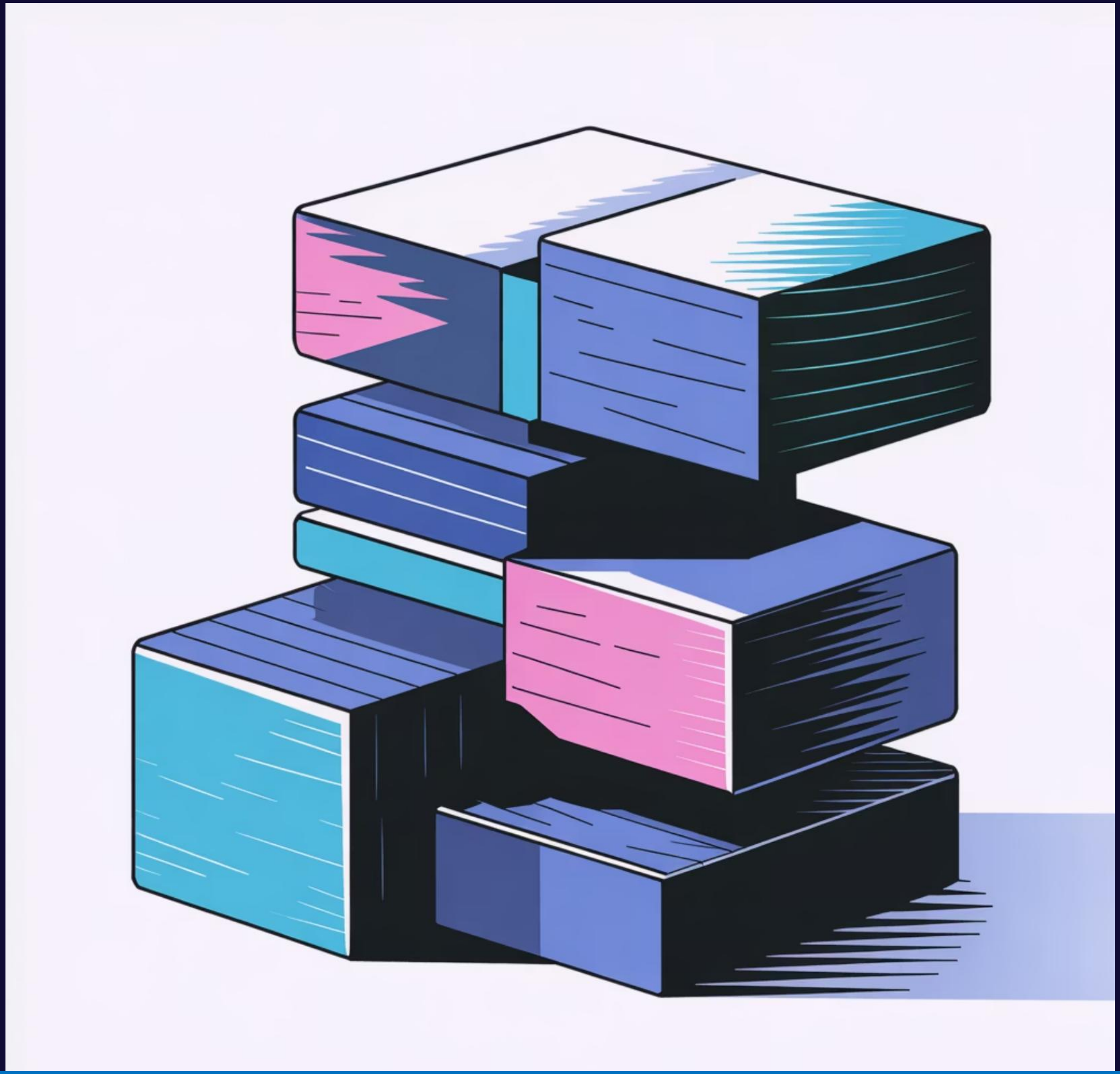
Operator appears after operands. Ideal for stack evaluation; parentheses unnecessary.

Stacks: core mechanics

Stack = LIFO structure. Two fundamental operations:

- PUSH: insert item on top
- POP: remove top item

Common uses: expression conversion/evaluation, function call frames, undo buffers.

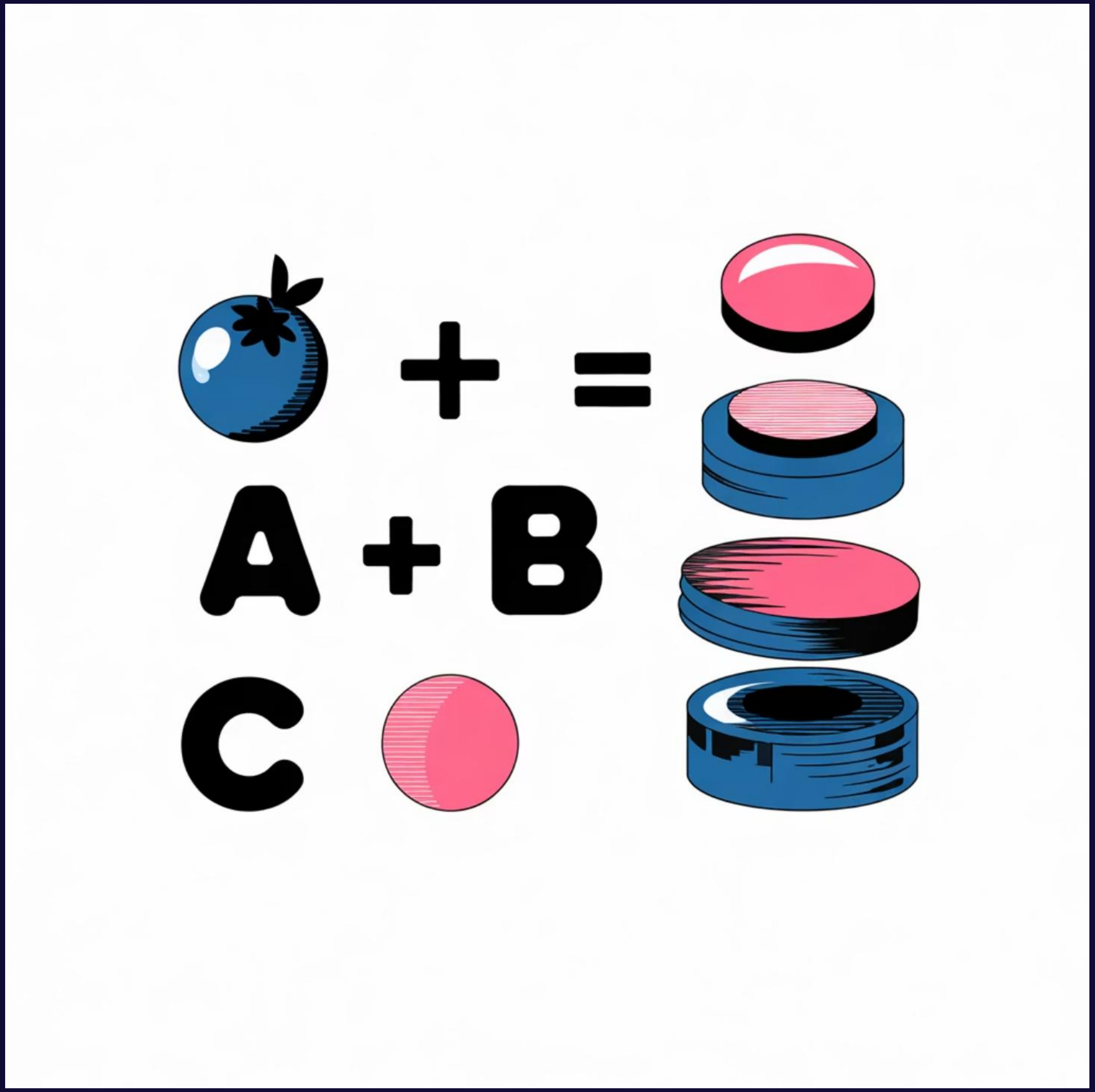


Convert Infix → Postfix (Shunting-yard idea)

Algorithm summary (concise):

1. Scan tokens left → right.
2. If operand → output immediately.
3. If operator → push to stack after popping higher-or-equal precedence operators (consider associativity).
4. Handle '(' by pushing, ')' by popping until '('.
5. At end, pop remaining operators to output.

Example: (A + B) * C → tokens: (A + B) * C → Postfix: A B + C *



Evaluate Postfix and Prefix using a Stack

Postfix evaluation

- Scan left → right.
- Operand → push numeric value.
- Operator → pop two operands (right operand popped first), apply op, push result.

Example: AB+C* with A=2, B=3, C=4 → steps: push 2,3 → + → push 5 → push 4 → * → result 20

Prefix evaluation

- Scan right → left.
- Operand → push.
- Operator → pop two operands, apply (first popped is left operand), push result.

Example: *+23 5 → compute (2+3)=5 then 5*5 = 25

Operator precedence & associativity (reference)

Operator	Precedence (higher → higher priority)	Associativity
^	3	Right → Left
* /	2	Left → Right
+ -	1	Left → Right

When converting, treat ^ as right-associative (so it does not get popped by equal precedence operators on the stack).

Applications & Advantages — practical view

Compiler internals

Abstract syntax trees and code generation rely on unambiguous expression forms and stack-based evaluation.

Calculators & interpreters

Postfix simplifies runtime evaluation: minimal parsing and fast execution.

Efficiency

Less memory overhead and deterministic evaluation order — beneficial for embedded systems.

Key takeaways & next steps

- **Polish notation** (prefix/postfix) removes parentheses and clarifies evaluation order.
- **Stacks** provide a simple, reliable mechanism for both conversion and evaluation.
- Practice: convert infix expressions by hand, then implement stack-based evaluators in code (C/C++).

Suggested exercises: 1) Convert and evaluate expressions with mixed operators and parentheses. 2) Implement infix → postfix converter and postfix evaluator. 3) Extend to support multi-digit numbers and variables.

Thank you — Questions?

Presented by Chanpura Keyur D., Chanpura Shivam M., Maheriya Hardik S., Wadhvana Shyam B.