stacks (LIFO: Last-in First-out)

2 Main operations: Push(x): Add a new element to the top of stack.

Pop(): removes and returns the top of stack.

Additional ops: peek(), isEmpty(), size(), clear(), . . .

$\downarrow$ top of stack (do not remove)

Implementation: Arrays - Elements occupy arr[0..size-1]

Top of stack index = size-1

Java: (old way): Stack<Integer> s = new Stack<>(),

(new way): Deque<Integer> s = new ArrayDeque<>(),

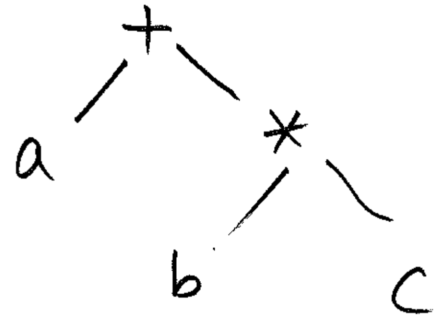
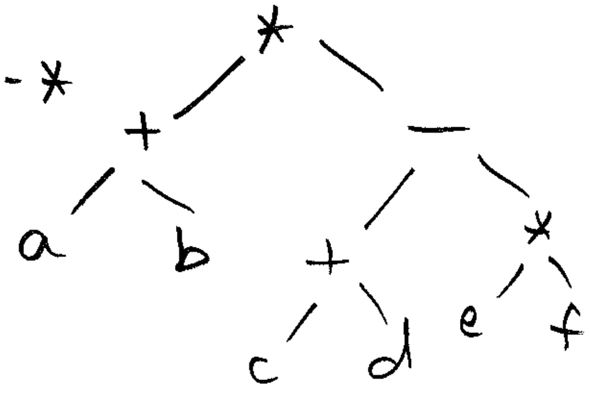
O(1) for push/pop. (O(1) amortized for push).

Applications of stacks:

1. Parsing of arithmetic expressions, Evaluation of exps.
2. Conversion of infix exp to postfix, Evaluation of postfix exp
3. Parsing of programming languages in compilers.
4. Memory management at run-time - functions - call. return
5. Depth-first search (DFS)
6. Maze creation

⋮

Arithmetic expressions:

Infix	Postfix	Expression Tree
$a + b * c$	$a\ b\ c * +$	
$(a+b)*(c+d-e*f)$	$a b + c d + e f * - *$	

Defs:

Infix expressions: Operator comes between operands.

- Precedence of ops, Parentheses required, Associativity of operators.

Precedence: hierarchy: exponentiation (^ , **)
Multiplication (*, /, %)
Addition (+, -)
⋮

Associativity: a op b op c = (a op b) op c
Left associative op

Right associative → = a op (b op c)

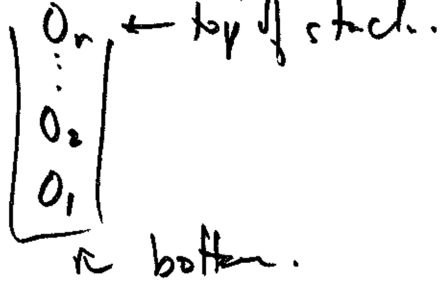
Postfix exp: Operator comes after operands.
Unambiguous - without parentheses, precedence, associativity, etc

Expression trees: binary trees.
internal nodes are operators
leaf nodes are operands.] - Parse tree of expression.

Parsing infix expressions:

(a) infix to postfix: Shunting Yard algorithm.

stack for operators —



→ { For left associative ops,
$\text{Prec}(O_{i+1}) > \text{Prec}(O_i)$

For right associative ops, $\text{Prec}(O_{i+1}) \geq \text{Prec}(O_i)$

Queue for Output.

Rules: Process input tokens one at a time.

(a) if token is "(" : Push "(" into stack

(b) if token = ")" : While top of stack is not "("
Pop from stack and add to Q
Pop '(' - discard

(c) if token = operator : if Prec(operator) > Prec(top of stack)
Push operator on stack
else While (Prec(OP) ≤ Prec(top of stack)): Pop from stack → add to Q

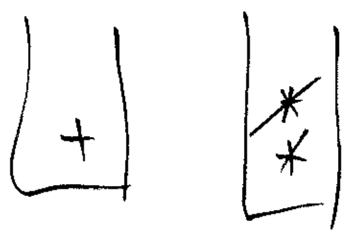
(d) if token = operand: add it to Q.

End: Pop from stack → add to Q until stack is empty.

Ex: Input: ~~a~~ ~~+~~ ~~b~~ ~~*~~ ~~c~~

Queue: a b c * +

stack:



Ex 2: ~~(~~~~a~~ ~~+~~ ~~b~~~~)~~ ~~*~~ ~~(~~~~c~~ ~~+~~ ~~d~~ ~~-~~ ~~e~~ ~~*~~ ~~f~~)

Queue: a b + c d + e f * - *

stack: