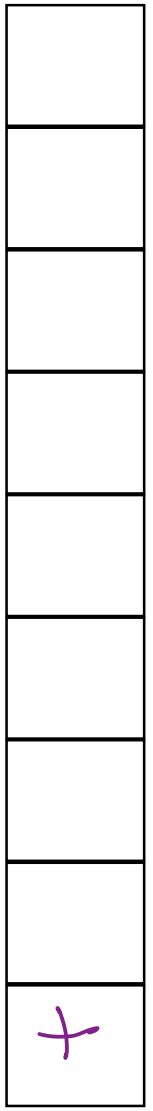
Data Structure & Algorithms

Nilesh Ghule

Infix to Postfix

Stack of operators

- 5 + 9 – 4 * (8 – 6 / 2) + 1 $ (7 – 3)



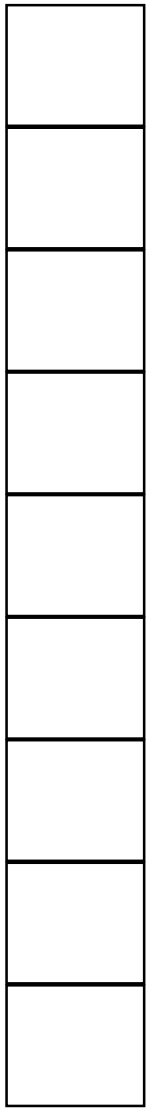
① infix expr process from left to right.
② if operand, append to postfix.
③ if operator, push on stack.
④ if topmost op on stack has priority greater or equal to current operator, then pop it and append to postfix expr.
⑤ when all syms from infix expr are done, pop all op from stack and append to postfix.
⑥ if opening (is found, push on stack.
⑦ if closing) is found, pop all op from stack and append to postfix until opening is found. Also discard (.

Infix to Prefix

stack of operators

- 5 + 9 – 4 * (8 – 6 / 2) + 1 $ (7 – 3)

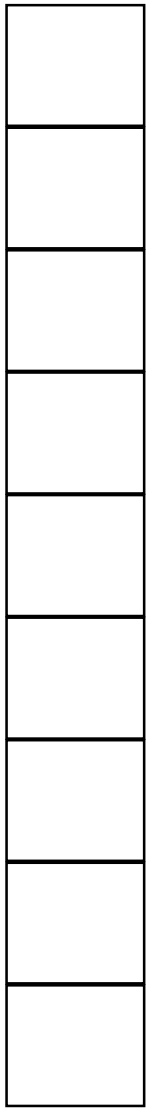
3 7 – 1 $ 2 6 / 8 – 4 * 9 5 + – +



Postfix Evaluation

Stack of operands

- 5 9 + 4 8 6 2 / - * - 1 7 3 - $ + *

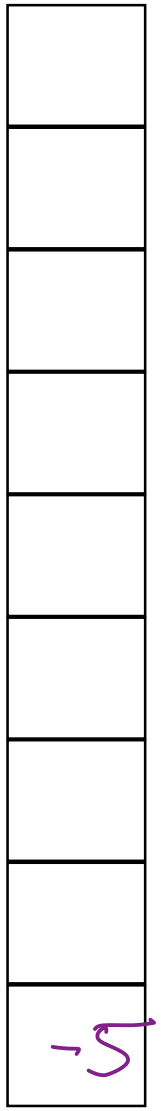


1. process postfix expr from left to right.
2. if operand, push it on stack.
3. if operator, pop two operands from stack calculate result & push result on stack.
   - first popped is second operand &
     second popped is first operand.
4. when all syms from postfix are done, pop final result from stack.

-5

Prefix Evaluation

- + - + 5 9 * 4 – 8 / 6 2 $ 1 – 7 3



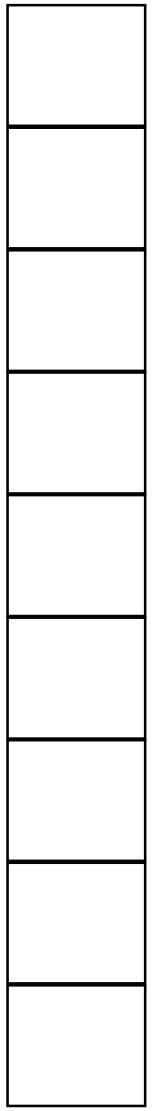
infix :

String[] syms = infix.split(" ");

Postfix to Infix

- While there are input symbol left
- Read the next symbol from input.
- If the symbol is an operand , Push it onto the stack.
- Otherwise, the symbol is an operator.
- If there are fewer than 2 values on the stack
- Show Error
- Else
- Pop the top 2 values from the stack.
- Put the operator, with the values as arguments and form a string.
- Encapsulate the resulted string with parenthesis.
- Push the resulted string back to stack.
- If there is only one value in the stack
- That value in the stack is the desired infix string.
- If there are more values in the stack
- Show Error

- a b c - + d e – f g – h + / *



Prefix to Postfix

- Read the Prefix expression in reverse order (from right to left)
- If the symbol is an operand, then push it onto the Stack
- If the symbol is an operator, then pop two operands from the Stack
- Create a string by concatenating the two operands and the operator after them.
- string = operand1 + operand2 + operator
- And push the resultant string back to Stack
- Repeat the above steps until end of Prefix expression.

- * + A B – C D

Parenthesis Balancing

- 5 + ([9 + 4] * (8 + { 6 / 2 }))

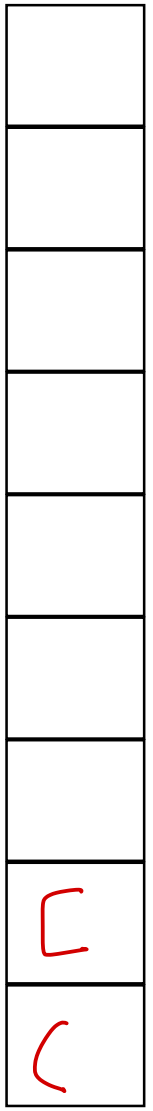
→ open
→ closing

0	1	2	3
(	[	{	<
)	]	}	>

① process from left to right.
② if opening, put on stack.
③ if closing, pop from stack,
compare index of closing & opening in respective arrays. if same, continue. else return false (stop).
④ when all symbols are finished, stack should be empty. then success.

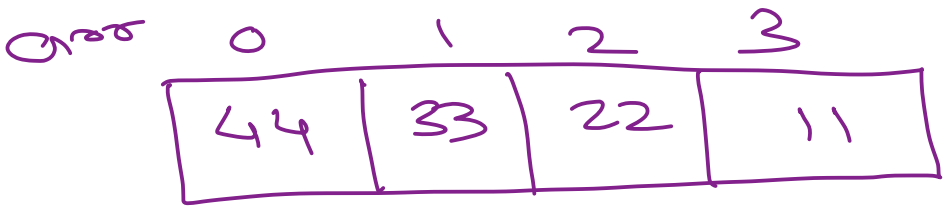
5 + ([9 + 4 })

} = 2 X
[= 1



Stack / Queue – Competitive Programming

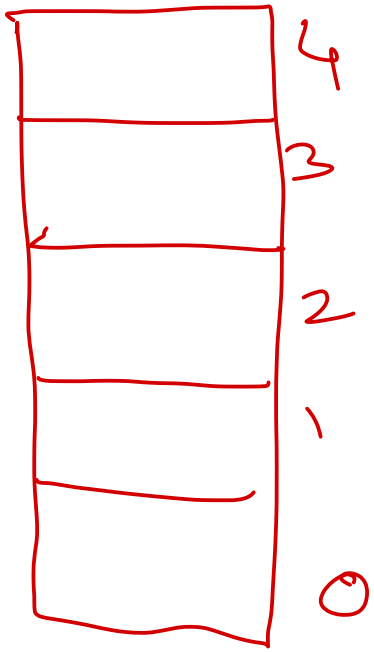
- Reverse array, string or linked list.




```
int arr[] = {11,22,33,44};
Stack<Integer> s = new Stack<>();

for(i=0; i<n; i++)
    s.push(arr[i]);

for(i=0; i<n; i++)
    arr[i] = s.pop();
```

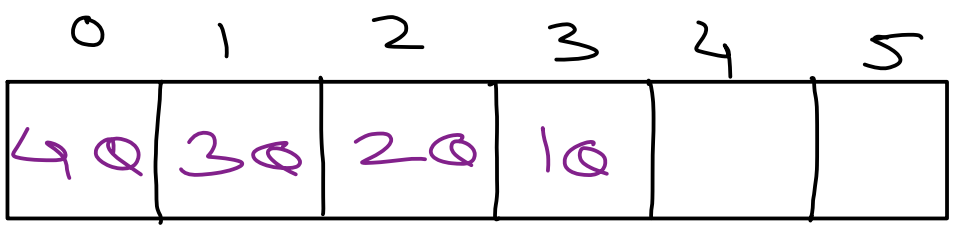


- Create stack using queue.
  - LIFO
  - FIFO

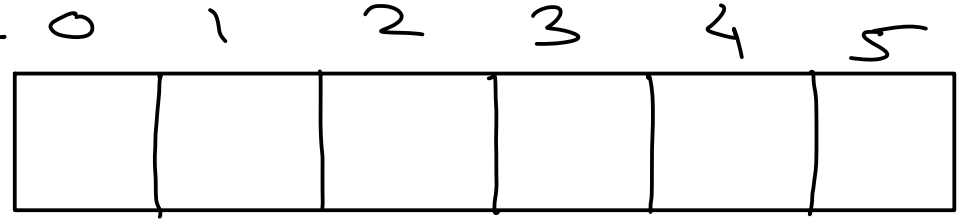
front

rear

q1



q2



temp

push:

1. pop all from main queue & add into temp queue.
2. push new ele in main queue
3. pop all from temp queue & add into main queue.

pop:

1. pop from main queue

Linked List - treasure hunt

- Linked List is list of items linked together.
- Each item in linked list is called as Node.
- Each node contains data and pointer/reference to the next node.
- Linked list is linear data structure.

Node

C → struct { data + next }

C++ → class { data + next }

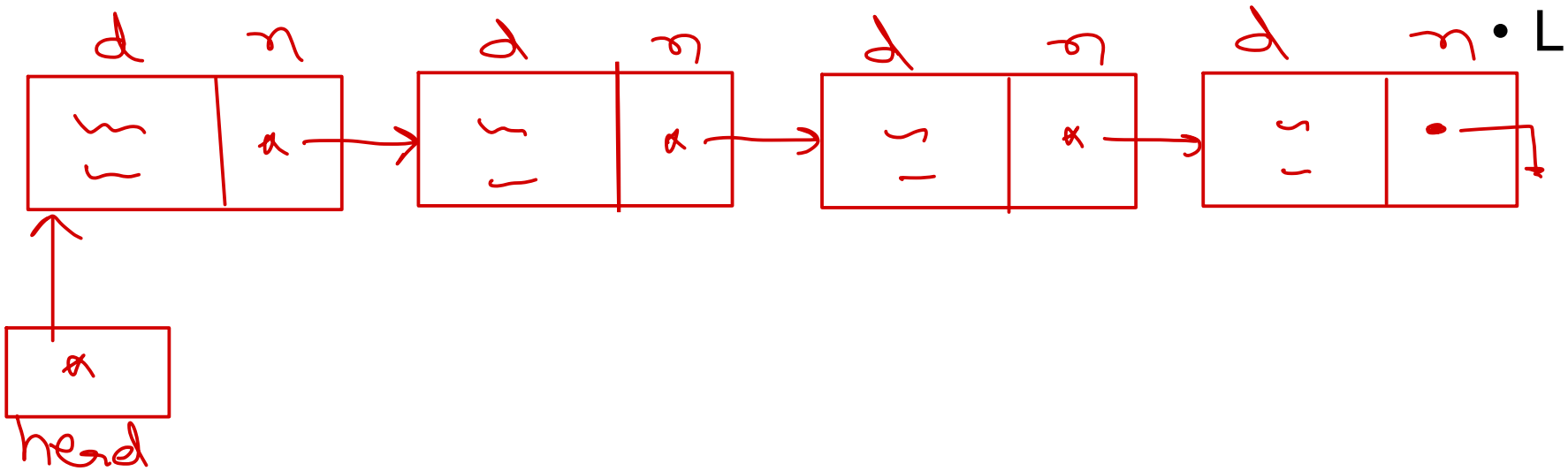
Java → class { data + next }

Python → class { data + next }

Linked List

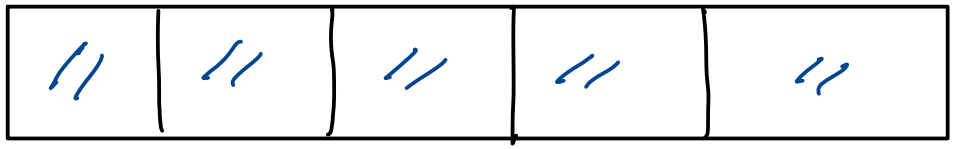
C → Node * head; – global

C++, Java, python → class { head }



- Linked list ADT
  - addFirst()
  - addLast()
  - addAtPos()
  - deleteFirst()
  - deleteLast()
  - deleteAsPos()
  - deleteAll()

array

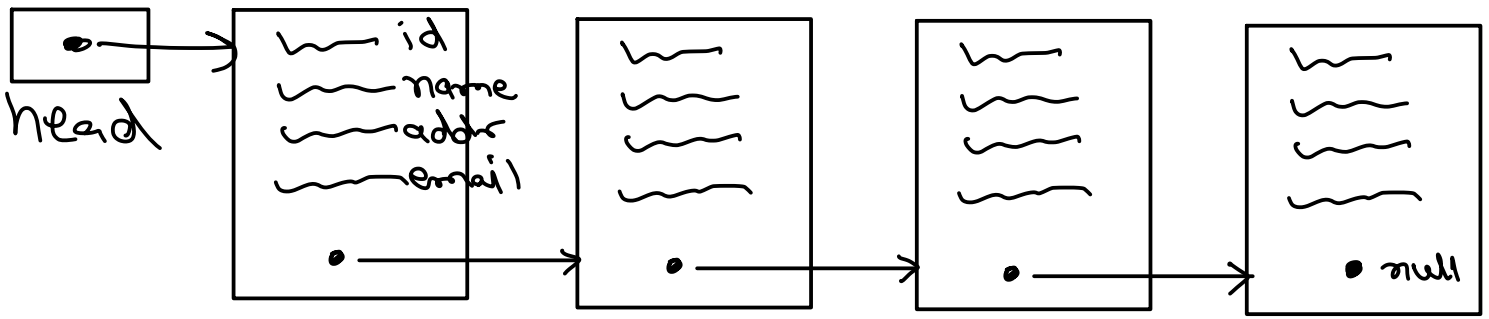


C/C++: int arr[5];
Java: int [] arr = new int[5];

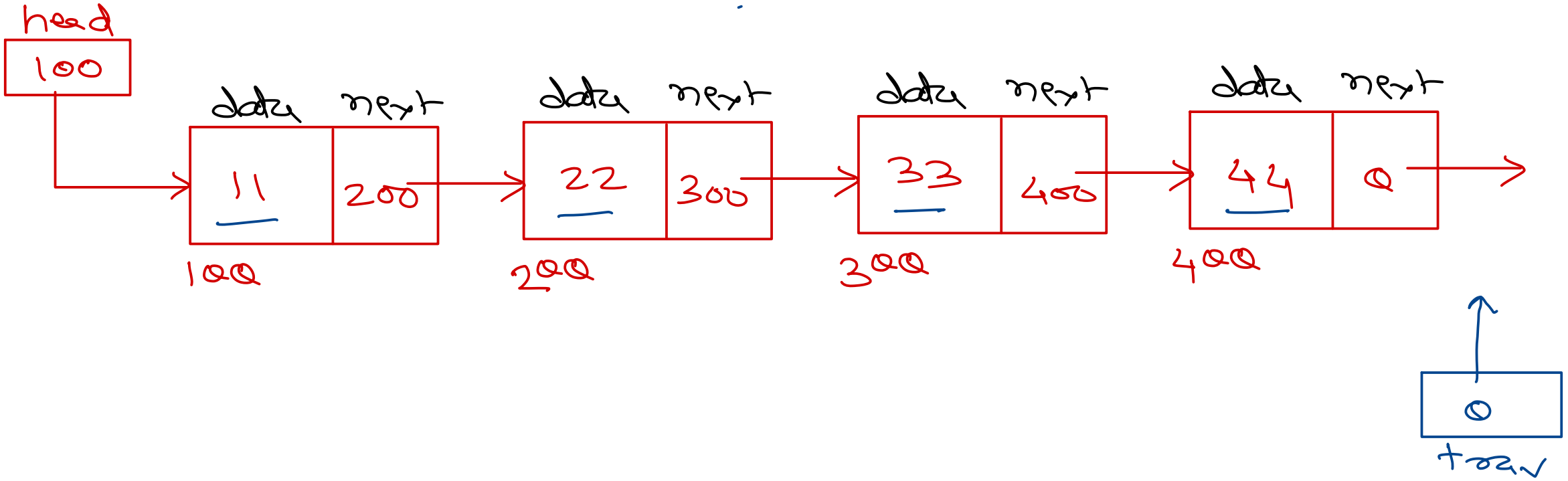
1. fixed size
2. no overheads
3. Contiguous memory
4. random access + sequential access.
5. insert/delete is not efficient.

linked list

1. grow/shrink dynamically
2. each item have extra pointer to next item.
3. non-contiguous mem.
4. sequential access only.
5. frequent add/deletion



Singly Linear Linked List → display()




```
trav = head;
while (trav != null)
{
    print (trav -> data);
    trav = trav.next;
}
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>