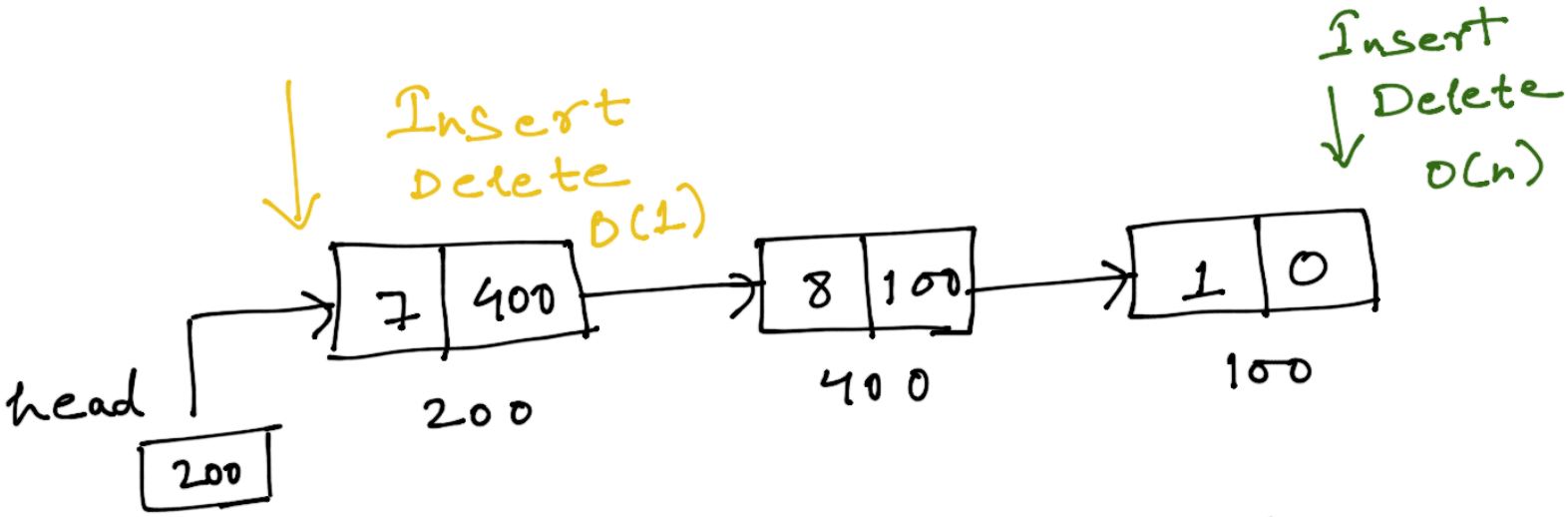
Implementation of Stack using Linked List

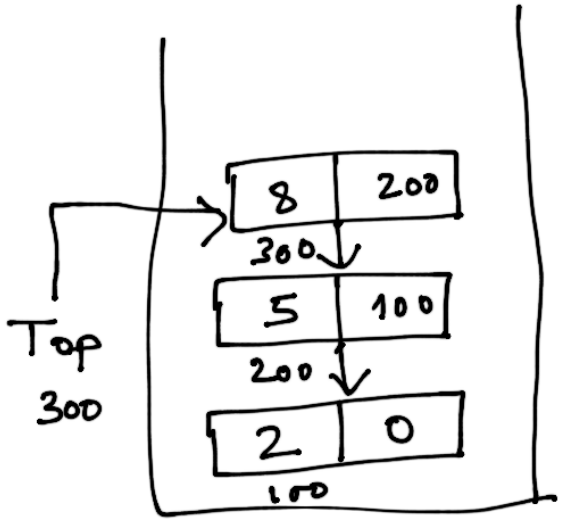


Stack

LIFO

push: O(1)

pop: O(1)

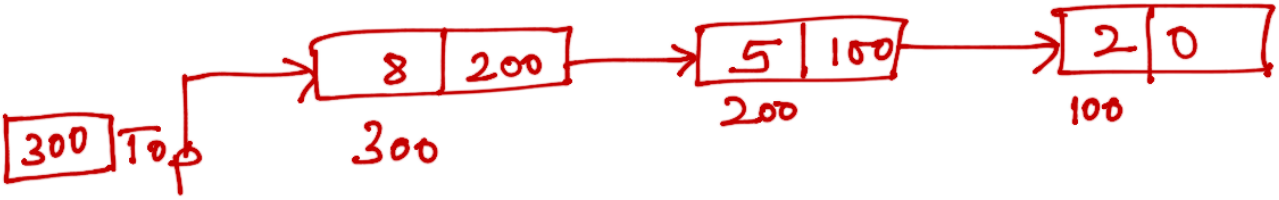


push(2)

push(5)

push(8)

push(10)



push (value)

Steps

1. Create a newNode and set data part = value
2. check whether stack is Empty (top == NULL)
3. If it is Empty, then set newNode -> next = NULL
4. If it is NOT Empty, then set newNode -> next = top
5. Then, set top = newNode

pop()

Steps

1. Check whether stack is Empty (top == NULL)
2. If it is Empty, then display "Stack underflow / stack is Empty"
3. If it is NOT Empty, then define a Node pointer 'temp' and set it to 'top'.
4. Then, set top = top → next
5. finally, delete 'temp'.

Infin / Prefin / Postfin

Infin : 5+1 which is Correct

5+1*6 = 36 —

= 11 —

Precendence

1) ( ), [ ], { }
2) ^ → R-L
3) * / → L-R
4) + - → L-R

Eg: 1+2*5 + 30/5

= 1 + 10 + 30/5

= 1 + 10 + 6

= 11+6

= 17

Eg: 2^2^3

= 2^8

= 256

NOT

2^2^3

→ 4^3

= 64 X

Prefin / Polish Notation

<operator> <operand> <operand>

$5+1 \Rightarrow +\ 5\ 1$

$a*b+c \Rightarrow *ab+c$

$\Rightarrow +*abc$

Postfix / Reverse Polish Notation

<operand> <operand> <operator>

$5+1 \Rightarrow 5\ 1\ +$

$a*b+c \Rightarrow ab*\ +\ c$

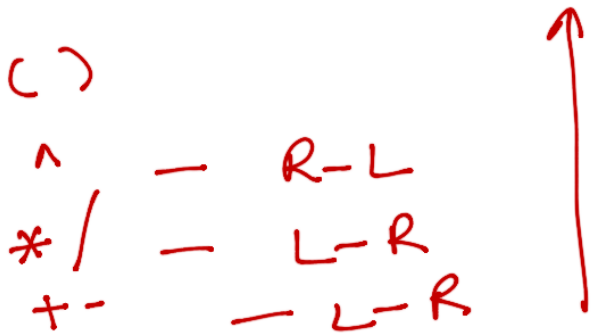
$\Rightarrow ab*\ c+$

→ Memory consumption is less for prefix / postfix

→ Easy for computers to evaluate

Infix to Postfix Using Stack

Precendence Rule



Eg: 1) A + B * C
= A + (B * C) (Scan 1)
= A + BC * (Scan 2)
= A BC * + postfix Exp

2) K + L - M * N + (O ^ P) * W (Scan 1)
= K + L - M * N + O P ^ * W (Scan 2)
= K + L - M N * + O P ^ * W (Scan 3)
= K + L - M N * + O P ^ W * (Scan 4)
= K L + - M N * + O P ^ W * (Scan 5)
= K L + M N * - + O P ^ W * (Scan 6)
= K L + M N * - O P ^ W * +

→ Using Stacks, we can evaluate this expression in just one scan

→ Once you reach the right end of the expression, we will get the equivalent postfix expression.

Eg: K + L − M

Input Exp	Stack	Postfix Exp
K		K
+	+	K
L	+	K L
−	−	K L +

('+' pops out of the stack bcoz of LR asso. through same precedence)

M	−	K L + M

Postfix Exp → K L + M −

Eg: K + L − M * N

Input Exp	Stack	Postfix Exp
K		K
+	+	K
L	+	K L
−	−	K L +
M	−	K L + M
*	− *	K L + M
N	− *	K L + M N

postfix Exp → K L + M N * −

Eg: $K+L-M*N+(O\wedge P)*N$

Input Exp	Stack	Postfix Exp
K		K
+	+	K
L	+	KL
−	−	KL+
M	−	KL+M
*	−*	KL+M
N	−*	KL+MN
+	+	KL+MN*−
(	+(	KL+MN*−
O	+(	KL+MN*−O
^	+(^	KL+MN*−O
P	+(^	KL+MN*−OP
)	+	KL+MN*−OP^
*	+*	KL+MN*−OP^
N	+*	KL+MN*−OP^N*+