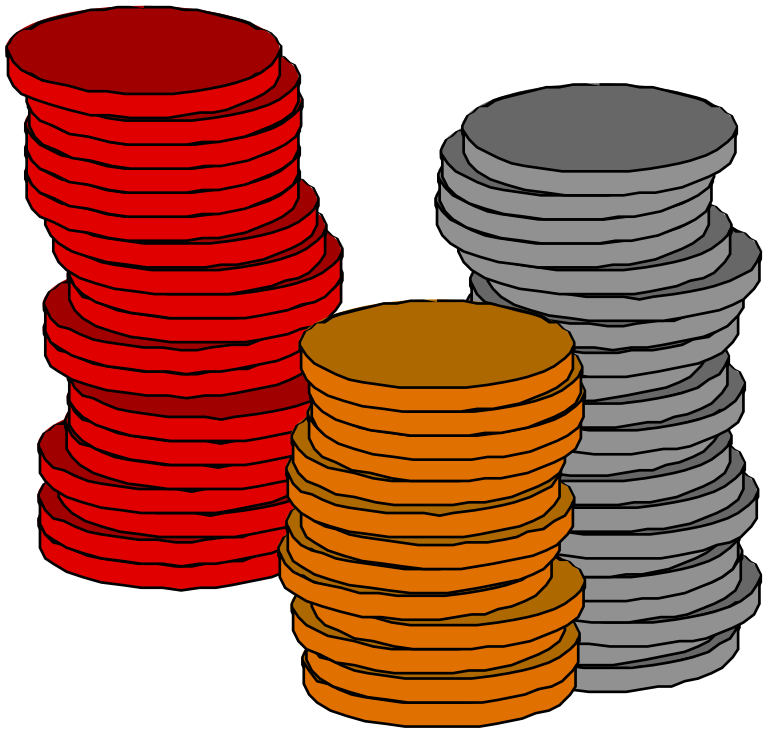
Data Structure & Algorithm
CS-102

Dr. Sambit Bakshi
Dept. of CSE, NIT Rourkela

Linear Data Structures

- There are certain frequent situations in computer science when one wants to restrict insertion and deletions so that they can take place only at the beginning or at the end not in the middle.
  - **Stack**
  - **Queue**

Stack



Stack

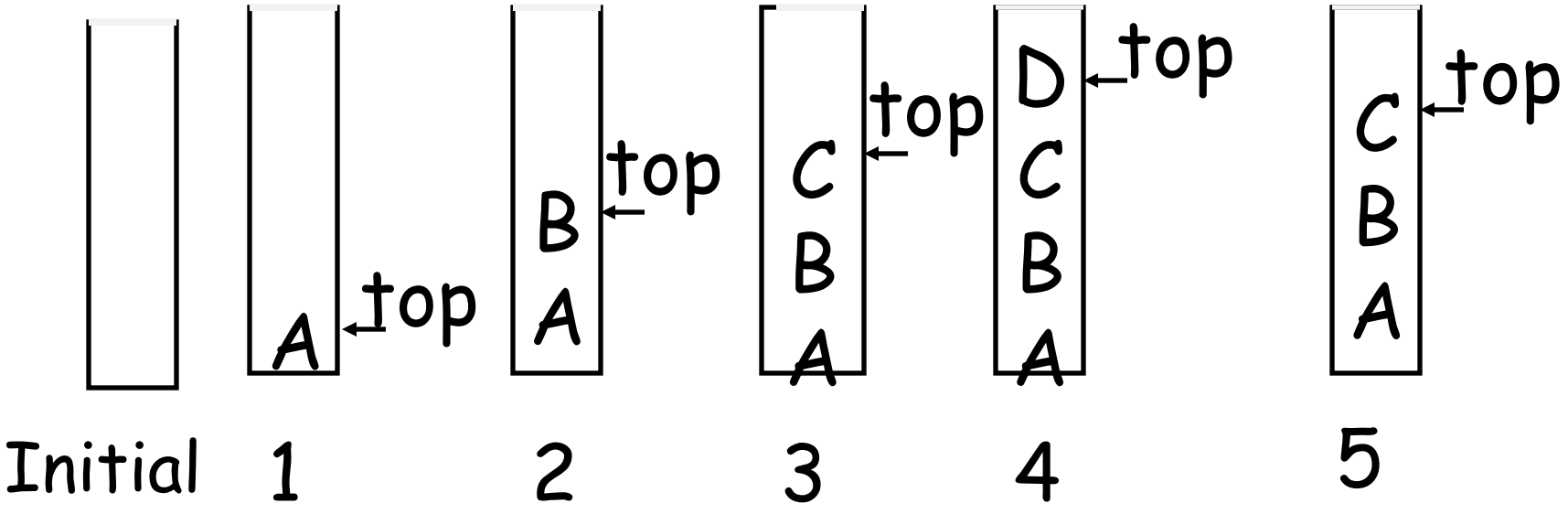
- A Stack is a list of elements in which an element may be inserted or deleted only at one end, call top of the Stack

- Two basic operations are associated with Stack
  - Push : Insert an element into a stack
  - Pop : Delete an element from a stack

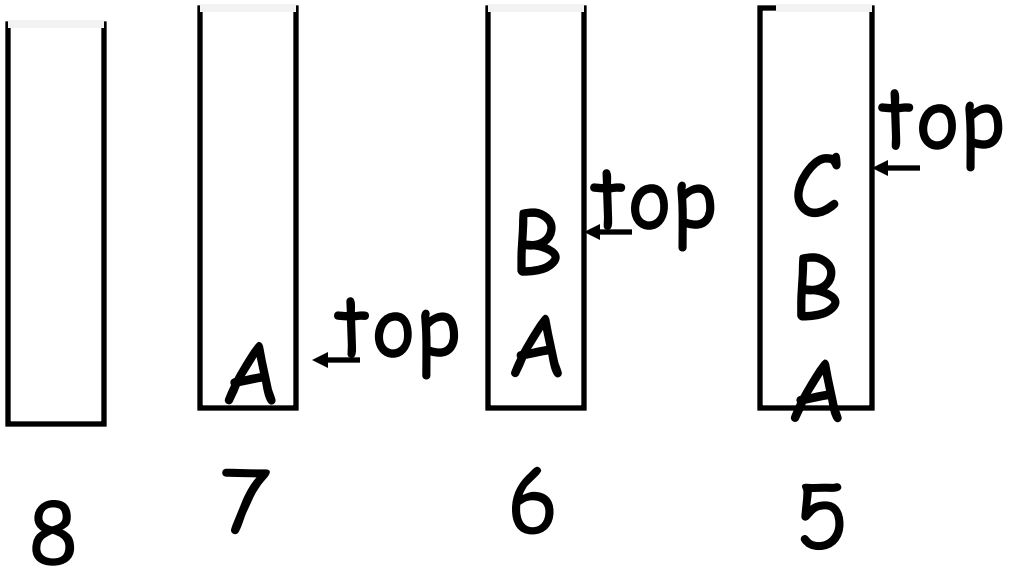
Stack

- Stores a set of elements in a particular order
- Stack principle: LAST IN FIRST OUT= LIFO
- It means: the last element inserted is the first one to be removed
- Which is the first element to pick up?

Last In First Out



Last In First Out



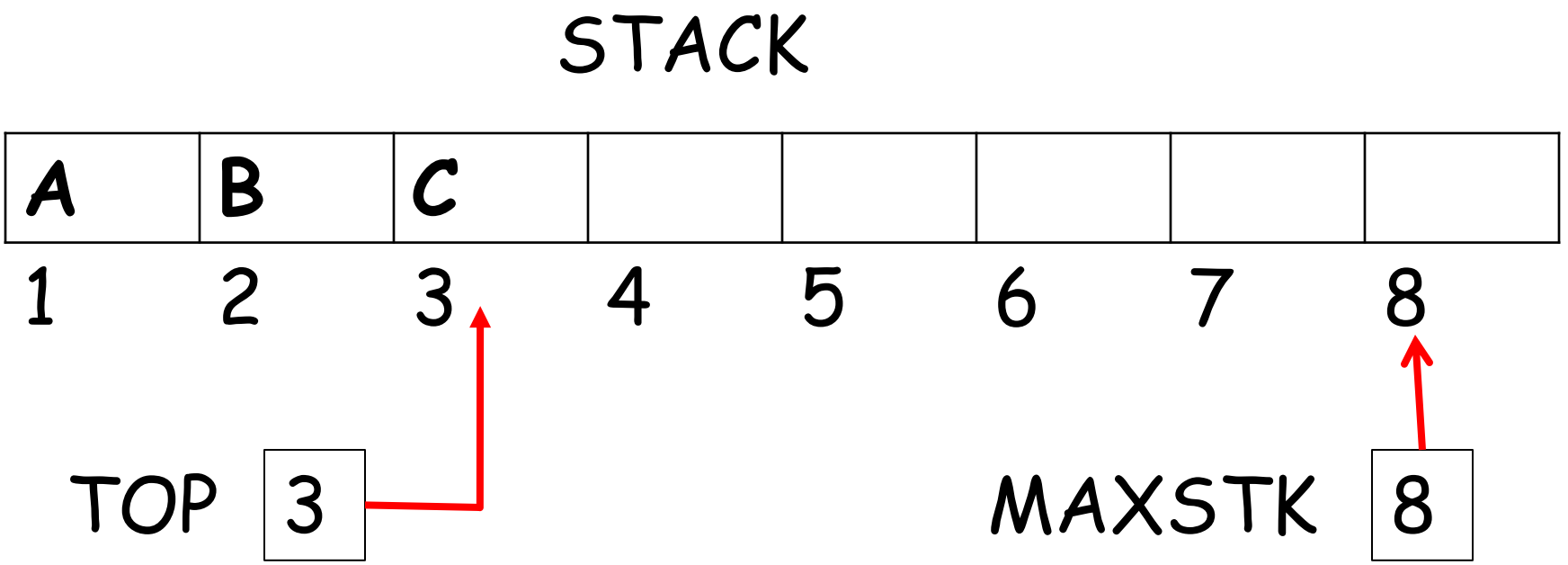
Representation of Stack

Stack can be represented in two different ways :

[1] Linear ARRAY (1-D)

[2] One-way Linked list

Array Representation of Stack



TOP = 0 will indicates that the stack is empty

PUSH Operation

Perform the following steps to PUSH an ITEM onto a Stack

[1] If TOP = MAXSTK, Then print: Overflow, Exit [Stack already filled]

[2] Set TOP = TOP + 1

[3] Set STACK[TOP] = ITEM [Insert Item into new TOP Position]

[4] Exit

POP Operation

Delete top element of STACK and assign it to the variable ITEM

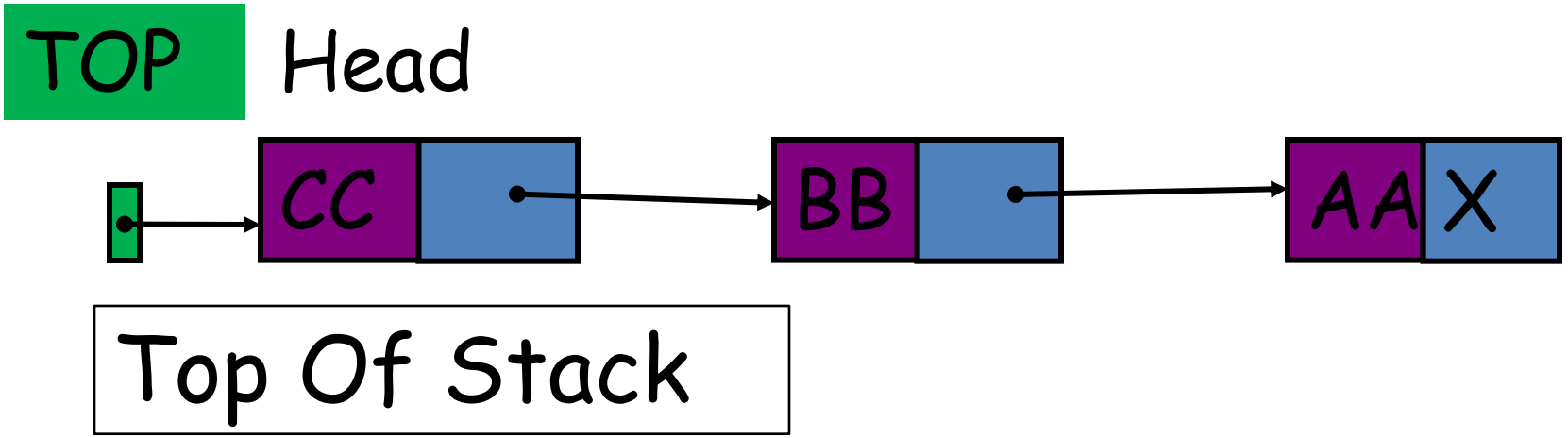
[1] If TOP = 0, Then print Underflow and Exit

[2] Set ITEM = STACK[TOP]

[3] Set TOP = TOP -1

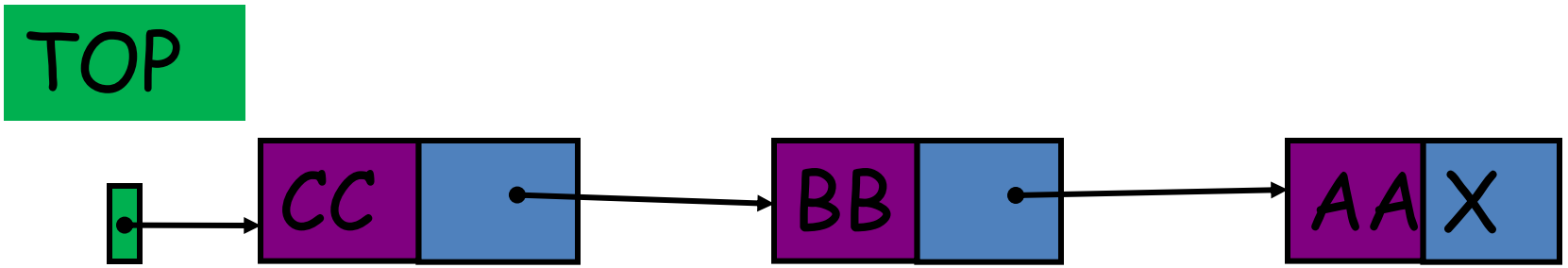
[4] Exit

Linked List Representation of Stack



PUSH Operation

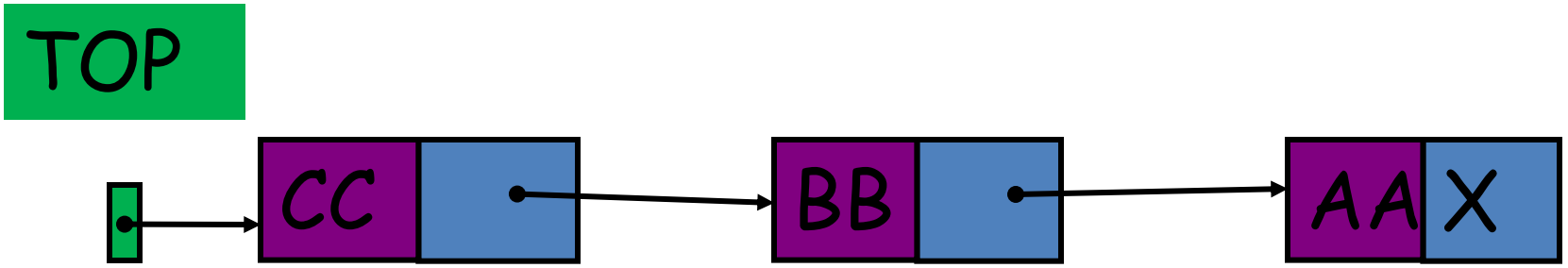
- Push operation into the stack is accomplished by inserting a node into the front of the list [**Insert it as the first node in the list**]



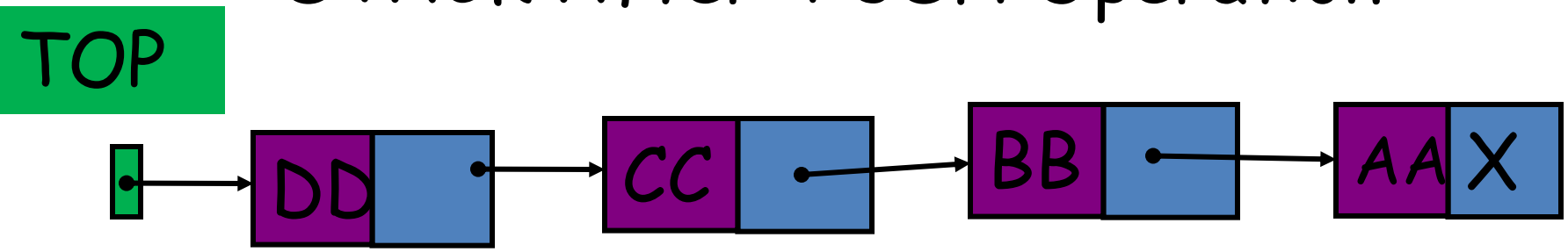
PUSH DD into STACK

PUSH Operation

STACK before PUSH Operation



STACK After PUSH Operation



PUSH Operation

[1] NEW->INFO = ITEM

[2] NEW->LINK = TOP

[3] TOP = NEW

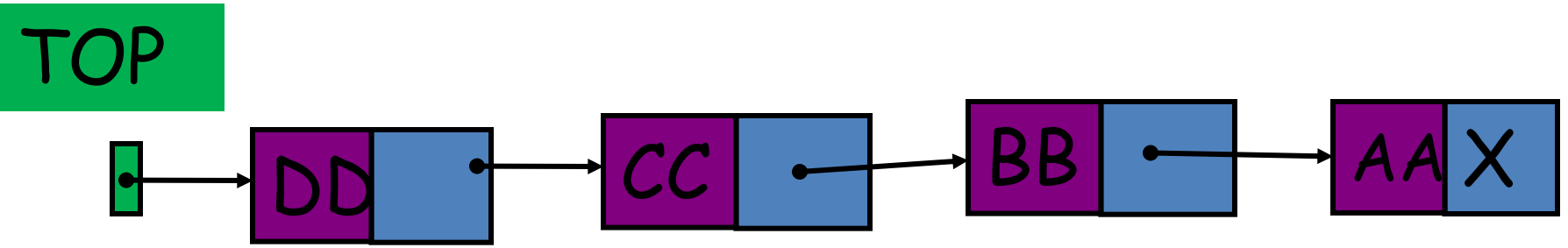
[4] Exit

POP Operation

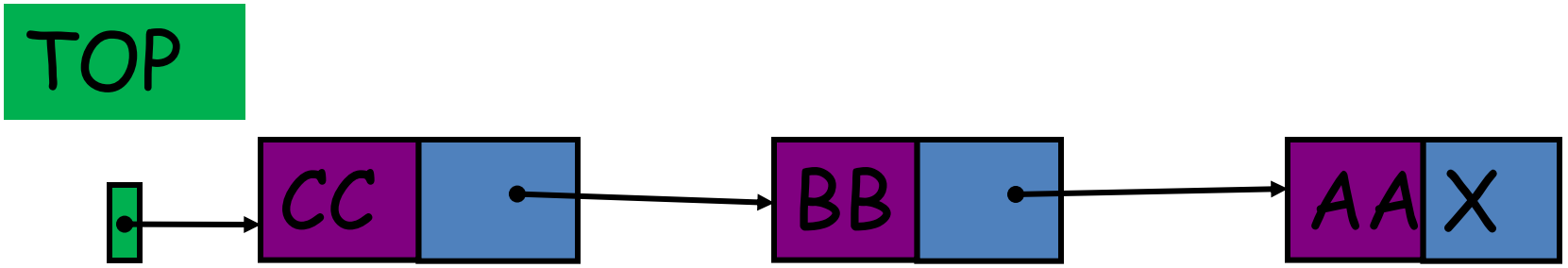
- POP operation is accomplished by deleting the node pointed to by the TOP pointer [**Delete the first node in the list**]

POP Operation

STACK before POP Operation



STACK After POP Operation



POP Operation

[1] IF TOP == NULL Then Write Underflow and Exit

[2] Set ITEM = TOP->INFO

[3] Set TOP = TOP->LINK

[4] Exit

Arithmetic Expression; Polish Notation

- Let **Q** be an arithmetic expression involving constant and operations

- Find the value of **Q** using reverse Polish (**Postfix**) Notation

Polish Notation

- Evaluate the following parenthesis-free arithmetic expression

2 î 3 + 5 * 2 î 2 – 12 / 6

Evaluate the exponentiation to obtain

8 + 5 * 4 – 12 /6

Evaluate Multiplication and Division

8 + 20 – 2

Evaluate Addition and Subtraction

26

Polish Notation

- **Infix notation** [Operator symbol is placed between two Operand]

  A + B , C – D , E * F , G /H

  (A + B) * C and A + (B*C)

- **Polish Notation** [Operator symbol is placed before its operand]

  +AB, -CD, *EF , /GH

Polish Notations are frequently called **Prefix**

Polish Notation

- Infix expression to Polish Notation

[ ] to indicate a partial translation

(A+B)*C = [+AB]*C = *+ABC

A+(B*C) = A+[*BC] = +A*BC

(A+B)/(C-D) = [+AB]/[-CD] = /+AB-CD

Polish Notation

- The fundamental property of Polish notation is that the order in which the operations are to be performed is completely determined by the positions of the operators and operand in the expression.
- **One never needs parenthesis when writing expression in Polish notations**

Reverse Polish Notation

- Operator symbol is placed after its two operand

  AB+, CD-, EF*, GC/

  (A+B)/(C-D) = [AB+]/[CD-] = AB+CD-/

- **One never needs parenthesis to determine the order of the operation in any arithmetic expression written in reverse Polish notation.**
- **Also known as Postfix notation**

- Computer usually evaluates an arithmetic expression written in infix notation in two steps:
- First Step: Converts the **expression to Postfix notation**
- Second Step: **Evaluates the Postfix expression.**

Evaluation of Postfix Expression

- Algorithm to find the **Value** of an arithmetic expression **P** Written in **Postfix**

[1] Add a right parenthesis ‘)” at the end of P. [This act as delimiter]

[2] Scan P from left to right and repeat Steps 3 and 4 for each element of P until the delimiter “)” is encountered

Evaluation of Postfix Expression

[3] If an operand is encountered, put it on STACK

[4] If an operator $\otimes$ is encountered, then

(a) Remove the two top elements of STACK, where A is the top element and B is the next-to-top element

(b) Evaluate $B \otimes A$

(c) Place the result of (b) on STACK

Evaluation of Postfix Expression

[5] Set **Value** equal to the top element of STACK

[6] Exit

Example

- Q = 5 * (6 + 2) - 12 / 4 [Infix]
- P = 5, 6, 2, + , *, 12, 4, /, - [Postfix]

- P:

5,	**6,**	**2,**	**+,**	***,**	**12,**	**4,**	**/,**	**-,**	**)**
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)

5, 6, 2, +, *, 12, 4, /, -,)

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

Symbol Scanned		STACK		
(1)	5	5 (T)		
(2)	6	5,	6 (T)	
(3)	2	5,	6,	2 (T)
(4)	+	5,	8 (T)	
(5)	*	40 (T)		
(6)	12	40,	12 (T)	
(7)	4	40,	12,	4 (T)
(8)	/	40,	3 (T)	
(9)	-	37 (T)		
(10)	)			

Infix to Postfix

- Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix notation, P

Infix to Postfix

- Levels of precedence

Operators	Precedence
()	Highest
↑	High
* , /	Mid
+ , -	Low

3 - 2 * 8

= -13 or 8 ??

Infix to Postfix

- Two types of associativity: Left to right, and right to left

Operators	Associativity
* , /	Left to right
+ , -	Left to right

3 * 5 / 3

= 5 or 3 ??

Infix to Postfix

- Refer to the complete chart of operators for knowing the precedence and associativity defined in C.

Infix to Postfix

[1] Push "(" onto STACK and ")" to the end of Q

[2] Scan Q from Left to Right and Repeat Steps 3 to 6 for each element of Q until the STACK is empty

[3] If an operand is encountered, add it to P

[4] If a left parenthesis is encountered, push it onto STACK

[5] If an operator $\otimes$ is encountered, then:

(a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has precedence as or higher precedence than $\otimes$.

(b) Add $\otimes$ to STACK

[6] If a right parenthesis is encountered, then

(a) Repeatedly pop from the STACK and add to P each operator (on top of STACK) until a left parenthesis is encountered.

(b) Remove the left parenthesis. [Do not add it to P]

[7] Exit

INFIX TO POSTFIX CONVERSION

Input: Infix notation of an expression **Output:** Postfix string (initially blank)

Read infix notation from left to right

Type of input symbol	**What to do**
1. Operand	Append into postfix string
2. Operator	
2.1 Current operator has equal or higher precedence than existing operator in TOS	push this new operator into stack
2.2. Current operator has lower precedence than existing operator in TOS	pop all elements in stack one by one and keep appending in postfix string until encountering an opening parenthesis or the stack is empty The current operator is pushed in stack
2.3. Current operator is opening parenthesis	push opening parenthesis into stack
2.4. Current operator is closing parenthesis	pop all operators from stack occurring before latest opening parenthesis and append into postfix string, then pop the latest opening parenthesis

Example

- Q : A + (B * C - (D / E ↑ F) * G) * H

A	+	(	B	*	C	-	(	D	/	E	↑	F	)	*	G	)	*	H	)
1	2	3	4	5	6	7	8	9											20

A	+	(	B	*	C	-	(	D	/	E	î	F	)	*	G	)	*	H	)
1	2	3	4	5	6	7	8	9											20

Symbol Scanned		STACK	Expression P
1	A	(	A
2	+	(+	A
3	(	(+ (	A
4	B	(+ (	A B
5	*	(+ (*	A B
6	C	(+ (*	A B C
7	-	(+ (-	A B C *
8	(	(+ (- (	A B C *

A	+	(	B	*	C	-	(	D	/	E	↑	F	)	*	G	)	*	H	)
1	2	3	4	5	6	7	8	9											20

Symbol Scanned	STACK	Expression P
8 (	(+ (- (	A B C *
9 D	(+ (- (	A B C * D
10 /	(+ (- (/	A B C * D
11 E	(+ (- (/	A B C * D E
12 ↑	(+ (- (/ ↑	A B C * D E
13 F	(+ (- (/ ↑	A B C * D E F
14)	(+ (-	A B C * D E F ↑ /
15 *	(+ (- *	A B C * D E F ↑ /

A	+	(	B	*	C	-	(	D	/	E	↑	F	)	*	G	)	*	H	)
1	2	3	4	5	6	7	8	9											20

Symbol Scanned	STACK	Expression P
15 *	(+ (- *	A B C * D E F ↑ /
16 G	(+ (- *	A B C * D E F ↑ / G
17)	(+	A B C * D E F ↑ / G * -
18 *	(+ *	A B C * D E F ↑ / G * -
19 H	(+ *	A B C * D E F ↑ / G * - H
20)		A B C * D E F ↑ / G * - H * +

The algorithm halts as stack is empty.

A	+	(	B	*	C	-	(	D	/	E	î	F	)	*	G	)	*	H	)
1	2	3	4	5	6	7	8	9											20

Symbol Scanned		STACK	Expression P
		(	
1	A	(	A
2	+	(+	A
3	(	(+ (	A
4	B	(+ (	A B
5	*	(+ (*	A B
6	C	(+ (*	A B C
7	-	(+ (-	A B C *
8	(	(+ (- (	A B C *

A	+	(	B	*	C	-	(	D	/	E	↑	F	)	*	G	)	*	H	)
1	2	3	4	5	6	7	8	9											20

Symbol Scanned		STACK	Expression P
8	(	(+ (- (	A B C *
9	D	(+ (- (	A B C * D
10	/	(+ (- (/	A B C * D
11	E	(+ (- (/	A B C * D E
12	↑	(+ (- (/ ↑	A B C * D E
13	F	(+ (- (/ ↑	A B C * D E F
14	)	(+ (-	A B C * D E F ↑ /
15	*	(+ (- *	A B C * D E F ↑ /

A	+	(	B	*	C	-	(	D	/	E	î	F	)	*	G	)	*	H	)
1	2	3	4	5	6	7	8	9											20

Symbol Scanned		STACK	Expression P
15	*	(+ (- *	A B C * D E F î /
16	G	(+ (- *	A B C * D E F î / G
17	)	(+	A B C * D E F î / G * -
18	*	(+ *	A B C * D E F î / G * -
19	H	(+ *	A B C * D E F î / G * - H
20	)		A B C * D E F î / G * - H * +

DO IT NOW!

(a)Convert this infix expression to postfix expression using stack (mention every step)

(2+3)↑(1+2)*5-6*20/4*(4↑4+4)-25

(b) Evaluate the postfix expression using stack.