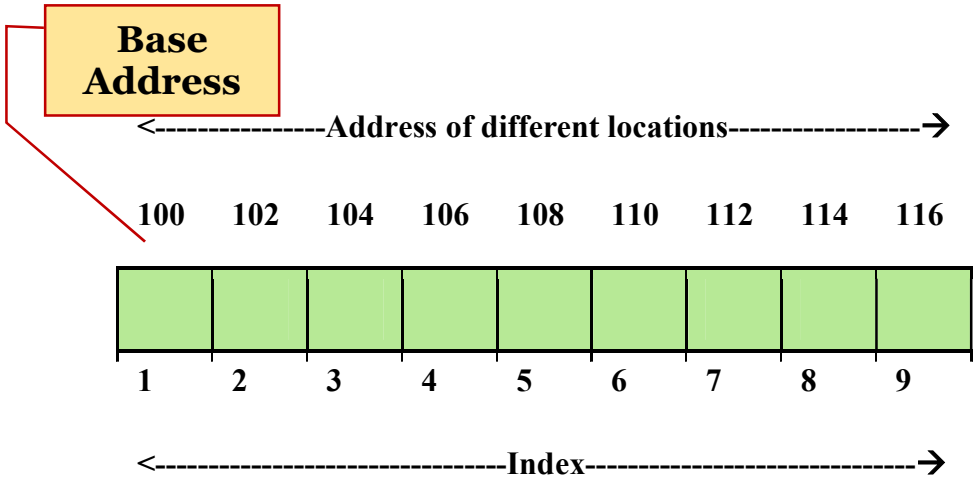
Module-2
(Concept of Array and Stack)

Array :

It is a linear data structure where the elements will store and process in a sequential manner.



- **The first location of the array is called as Lower Bound of the Array.**
- **The last location of the array is called as Upper Bound of the Array.**
- **The Address of the first location of the Array is called as Base address of the Array.**

Overflow condition :

- **If the structure is full of elements, and after that if we will try to insert an element into the structure, then such condition is called as overflow condition.**

Underflow condition :

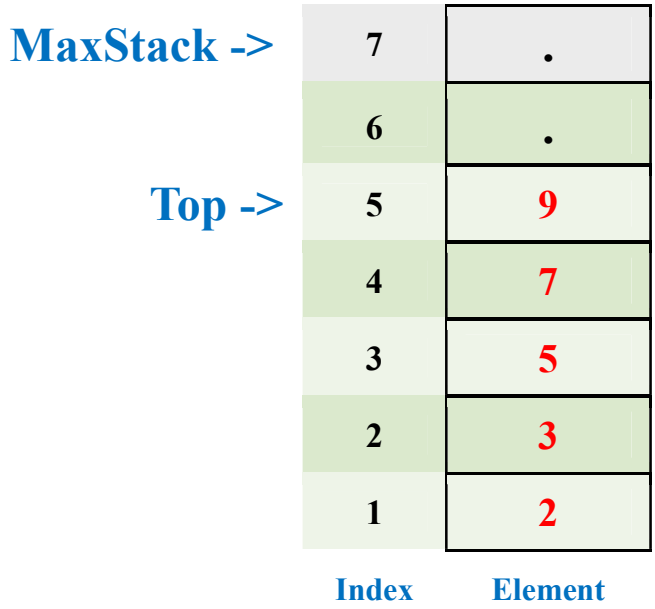
- **If the structure is totally empty, and after that if we will try to delete an element from the structure, then such condition is called as underflow condition.**

Stack :

It is a linear array where both the insertion and deletion operation will be performed at a single end i.e. called as TOP end of the stack.

Top: **It is a pointer which points to the location of top element in the Stack.**

Max_Stack: **It is a pointer which points the size of the Stack.**



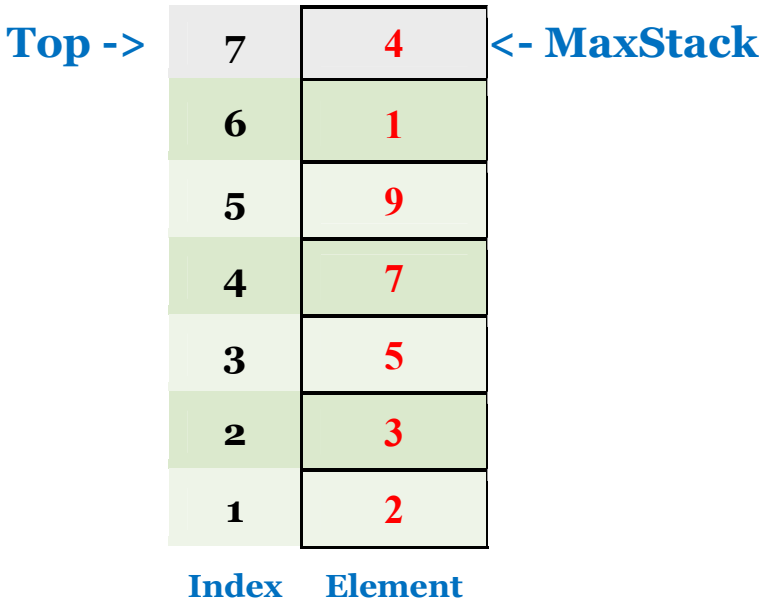
In Stack there are two operations can be performed named as Push and Pop operation

- ⇨ **The insertion operation into the stack is called as PUSH operation. Similarly, the deletion operation from the stack is called as POP operation.**

Though in stack both insertion and deletion operation performed at one end only, hence we can say the Stack follows LIFO procedure in memory.

LIFO means Last In First Out

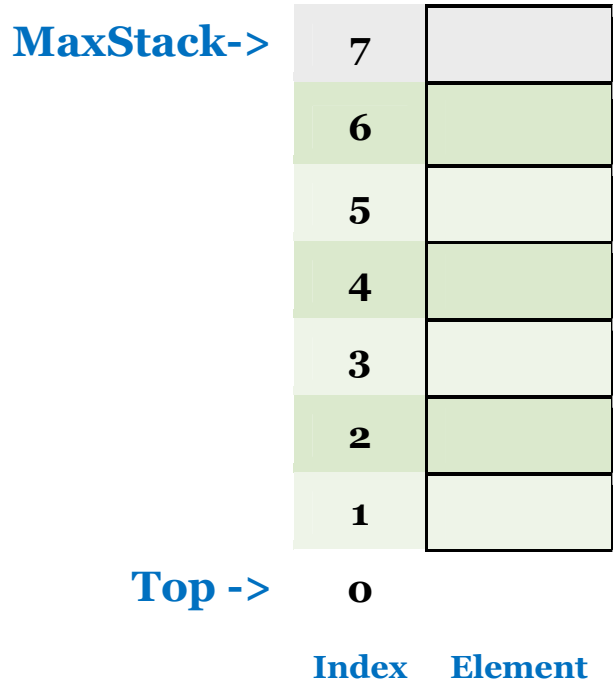
Overflow condition of Stack :



In algorithm if Top = MAXSTACK, then such condition is called as overflow condition of a stack.

Underflow condition of a Stack :

In algorithm if Top = 0, then such condition is called as underflow condition of a stack.



Algorithm for Push Operation:

Push (Stack, item, top, MAXStack)

S1: Input : item
S2: if (top = MAXStack), then

Print : 'Overflow'

return

[end of if]

S3: set top := top+1
S4: set Stack[top] := item
S5: exit

Algorithm for Pop Operation:

Pop (Stack, item, top)
S1: if(top=0), then

Print : 'UNDERFLOW'

Return

[end of if]

S2: set item:=Stack[top]

Set top:=top-1

S3: exit

Program for push, Pop, and display operation in Stack

```
import java.util.*;
public class Trial
{
   int stack[];
   int top;
   int maxstack;
   Trial(int n)
   {
      stack=new int[n];
      top=-1;
      maxstack=n-1;
   }
   void push(int item)
   {
      if(top==maxstack)
      {
         System.out.println("OVERFLOW");
         return;
      }
      else
      {
         top=top+1;
         stack[top]=item;
      }
   }
```

```
void del()
{
   int item;
   if(top==-1)
   {
      System.out.println("UNDERFLOW");
      return;
   }
   item=stack[top];
   top=top-1;
   System.out.println("Deleted item is :" + item);
}
void display()
{
   System.out.println (" The elements of the Stack are : ");
   for(int i=0;i<=top;i++)
   {
      System.out.println(stack[i]);
   }
}
public static void main(String args[])
{
     int n;
     System.out.println("Enter the size of the Stack");
     Scanner sc = new Scanner(System.in);
     n=sc.nextInt();
     Trial t1=new Trial(n);
     System.out.println("Enter some elements into the Stack");
     t1.push(sc.nextInt());
```

```
            t1.push(sc.nextInt());
            t1.push(sc.nextInt());
            t1.push(sc.nextInt());
            t1.push(sc.nextInt());
            t1.display();
            t1.del();
            t1.del();
            t1.del();
            t1.del();
        }
}
```

Application of Stack:

In our day to day life the concept of Stack is implemented in various places such as….

1) **To convert an Infix expression to it's equivalent Postfix expression.**
2) **To evaluate a Postfix expression**
3) **To traverse a Graph by using DFS procedure etc..**

Infix Expression: **In an expression if the operator symbol will use in between two operands then that expression is called as Infix expression. e.g. A + B**

Prefix Expression: **In an expression if the operator symbol will use before the operands then that expression is called as Prefix expression. e.g. + A B**

Postfix Expression: **In an expression if the operator symbol will use after the operands then that expression is called as Postfix expression. e.g. A B +**

1) **Convert the Infix expression "a * b + c / d ^ e – f " into it's equivalent Prefix and Postfix expression.**

Conversion into Prefix Expression

S1: a * b + c / d ^ e – f

S2: a * b + c / (^d e) – f

S3: (*a b) + c / (^d e) – f

S4: (*a b) + (/ c^d e) – f

S5: (+ *a b / c ^ d e) – f

S6: - +*a b / c ^ d e f

Conversion into Postfix Expression

S1: a * b + c / d ^ e – f

S2: a * b + c / (d e ^) – f

S3: (a b*) + c / (d e ^) – f

S4: (a b*) + (c d e ^ /) – f

S5: (a b* c d e ^ / +) – f

S6: a b* c d e ^ / + f -

Procedure to convert an Infix expression to Postfix expression using STACK

S1: Add "(" at the beginning and ")" at the end of the infix Expression

S2: If an operand will encounter, then add that operand into postfix expression

S3: If an operator will encounter, then compare the priority of the infix operator with the Stack operator

- **(a)** If the stack operator is a higher priority operator then, the stack operator is need to be poped out and add it to the postfix expression and then push the infix operator into the stack.
- **(b)** If the stack operator is a lower priority operator then, simply push the infix operator into the stack.
- **(c)** If there is no stack operator then, simply push the infix operator into the stack.

S4: If a "(" will encounter then push it into the stack

S5: If a ")" will encounter, pop all the operators from the stack until a "(" will encounter in the stack, and then add all the operators into the postfix expression.

S6: Continue Step 2 to Step 5 until Stack is not empty.

Question:

Convert the following infix expression into postfix expression:

A+B^(C+D)-E*F+G

Infix Symbol	Operator Stack	Postfix Symbol
(	(	
A	(	A
+	(+	A
B	(+	A B
^	(+ ^	A B
(	(+ ^ (	A B
C	(+ ^ (	A B C
+	(+ ^ (+	A B C
D	(+ ^ (+	A B C D
)	(+ ^	A B C D +
-	(+ -	A B C D + ^
E	(+ -	A B C D + ^ E
*	(+ - *	A B C D + ^ E
F	(+ - *	A B C D + ^ E F
+	(+ - +	A B C D + ^ E F *
G	(+ - +	A B C D + ^ E F * G
)	Empty	A B C D + ^ E F * G + - +

Procedure to evaluate to a Postfix expression using STACK

S1: If an operand will encounter, then push it into the stack

S2: If an operator Ⓟ will encounter then,

- (a) Pop two elements (i.e. operands) from the stack
- (b) Perform an operation using the two popped operands with the operator Ⓟ
  Result = Top2 Ⓟ Top1
- (c) Push the result of the operation again into the stack

S3: Continue Step 1 and Step 2 for each element in the postfix expression

S4: Res = Stack [top]

Question:

Explain the procedure to evaluate postfix expression. 7 3 4 + - 2 4 5 / + * 6 / 7 +

							5								
		4				4	4	0							
	3	3	7		2	2	2	2	2		6		7		
7	7	7	7	0	0	0	0	0	0	0	0	0	0	7	
7	3	4	+	-	2	4	5	/	+	*	6	/	7	+	7