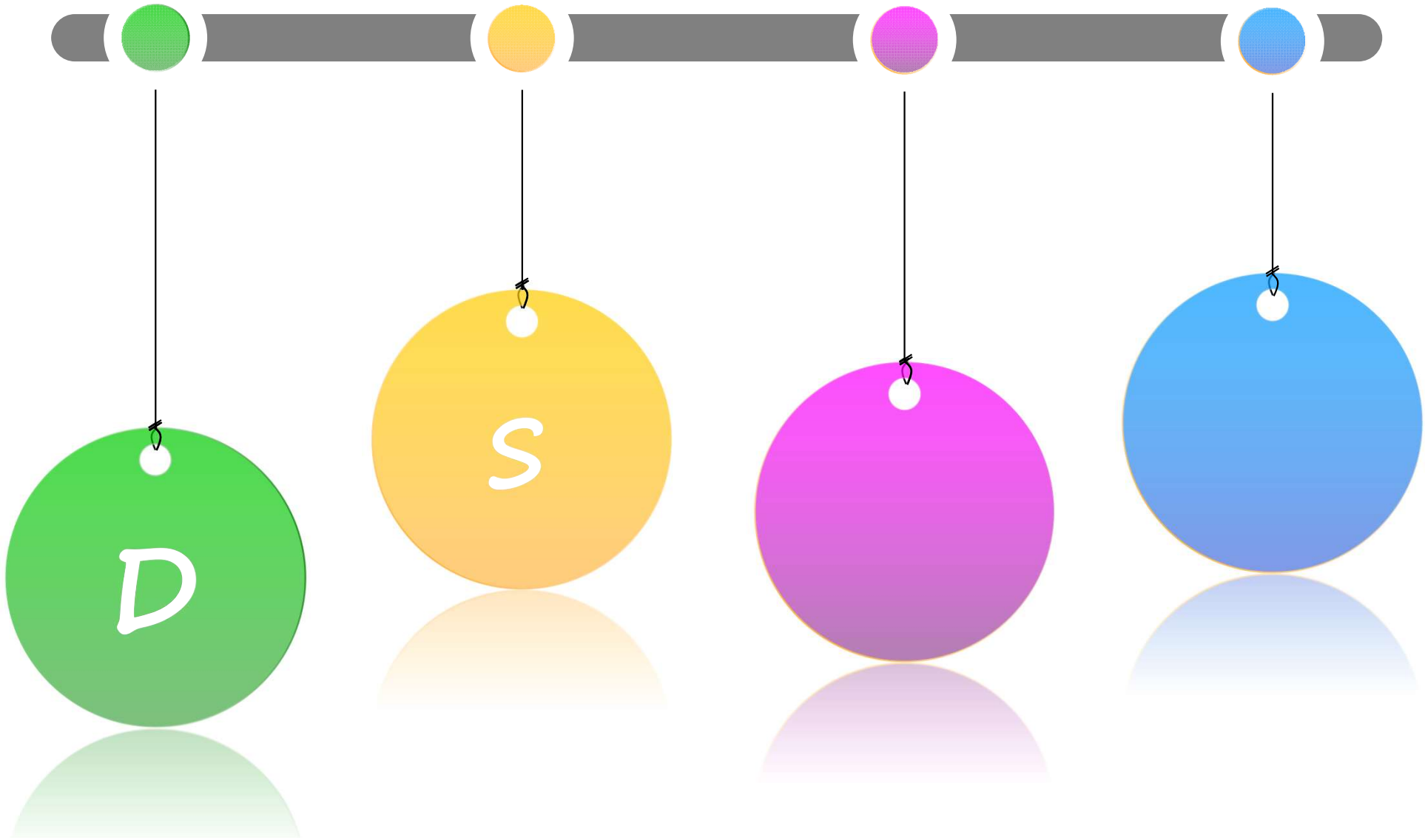

D
S

01 TOPIC — Stack: Definition

02 TOPIC — Array and Linked List representation of Stacks

03 TOPIC — Operations on Stack using Array.

04 TOPIC — Operations on Stack using Linked List.

05 TOPIC — Applications of Stacks: Infix to Postfix conversion

06 TOPIC — Evaluation of Postfix Expression.

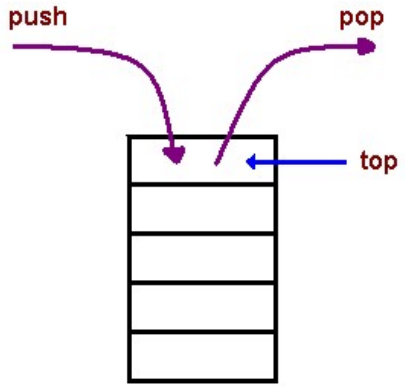
Stack

Definition of Stack:

i. A Stack is a **linear data structure** that follows the **LIFO** (Last-In-First-Out) principle.

ii. Stack is a data structure in which **items can be inserted and removed from the same end.**

- The **last item inserted into stack**, is the the **first item to be taken out from the stack**.
- In short its also called **Last in First out** [LIFO].
- LIFO stands for Last-in-first-out. Here, **the element which is inserted last, is accessed first.**
- Open end of the stack is called **Top**, From this **end item can be inserted and removed.**



Stack

- There are two ways to implement a stack:

i. **Using array**

ii. **Using linked list**

Implement of stack Using array:

Syntax:

```
#define Macro_Name  Value

Datatype  Array_Name[Macro_Name ];

Int Var_Name=-1;
```

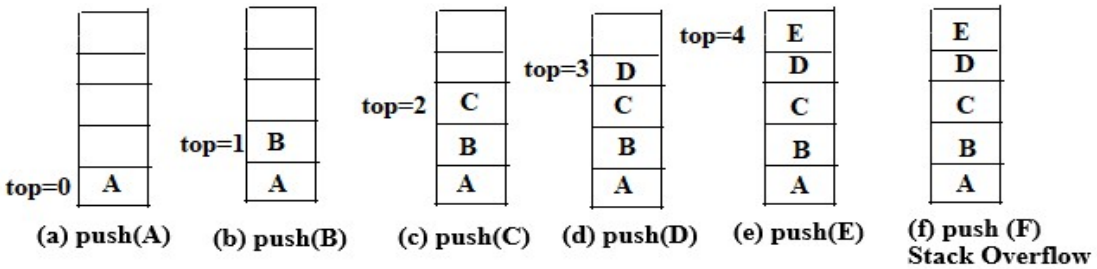
Ex:

```
#define Maxsize 10

int  Stack[Maxsize];

int top=-1;
```

Basic operations of Stack

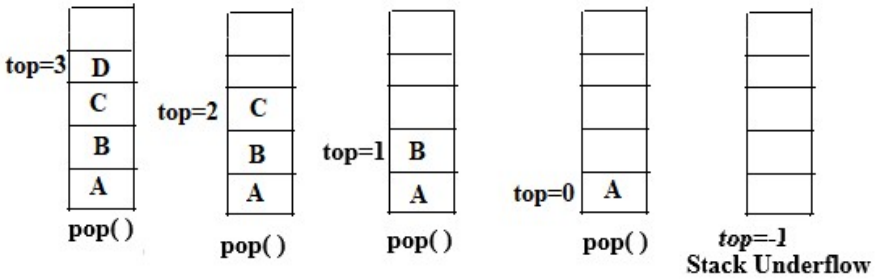
i.push():

- ✓ When we **insert an element in a stack** then the operation is known as a push.
- ✓ If the stack is full then the **overflow** condition occurs.



ii.pop():

- ✓ When we **delete an element from the stack**, the operation is known as a pop.
- ✓ If the stack is empty means that no element exists in the stack, this state is known as an **underflow.**



Basic operations of Stack

iii. isEmpty():

- ✓ It determines whether **the stack is empty or not.**

iv. isFull():

- ✓ It determines whether the **stack is full or not.**

v. peek():

- ✓ It Print the **top most element from the stack.**

Vi .display():

- ✓ It prints **all the elements available** in the stack.

i.Push Operation

- The process of putting a new data element into stack is known as a Push Operation.
- Push operation **involves a series of steps.**

Step 1 : **Checks** if the **stack is full.**

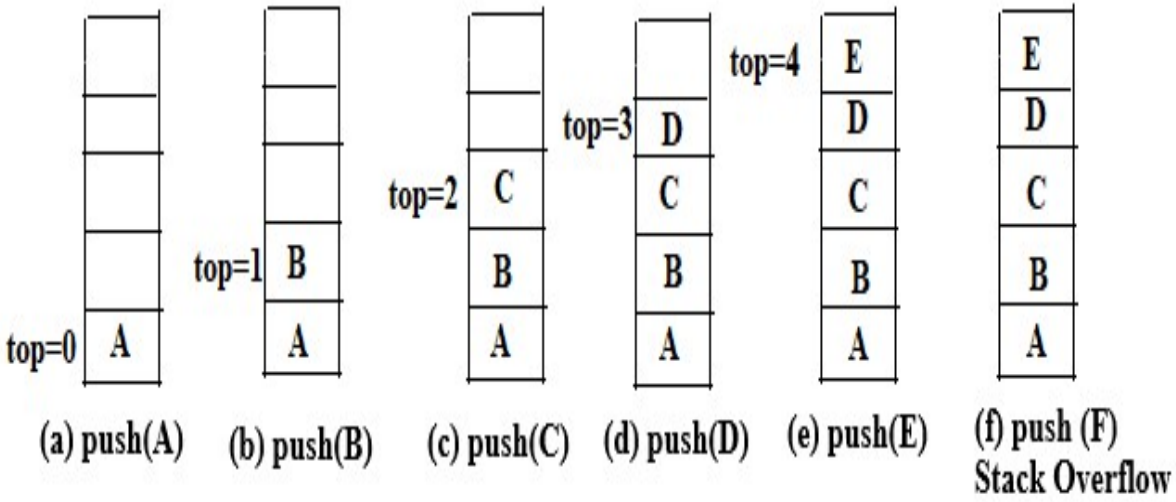
Step 2 : If the stack is full, **produces an Overflow message** and exit.

Step 3 : If the stack is **not full**, **increments top to point next empty space**.

Step 4 : Adds **data element to the stack location**, where top is pointing.

Push Operation algorithm

```
void push(int  ele)
begin
      if(top==Maxsize-1)
      {
      printf("Stack is full.\n");
      }
      else
      {
                  top = top + 1;
                  stack[top] = ele;
      }
 end
```



Pop Operation

- Removing the element from the stack known as a Pop Operation.
- In an array implementation of pop() operation, the **data element is not actually removed**, instead **top is decremented to a lower position** in the stack.
- Pop operation involves a series of steps:

Step 1 : **Checks** if the **stack is empty.**

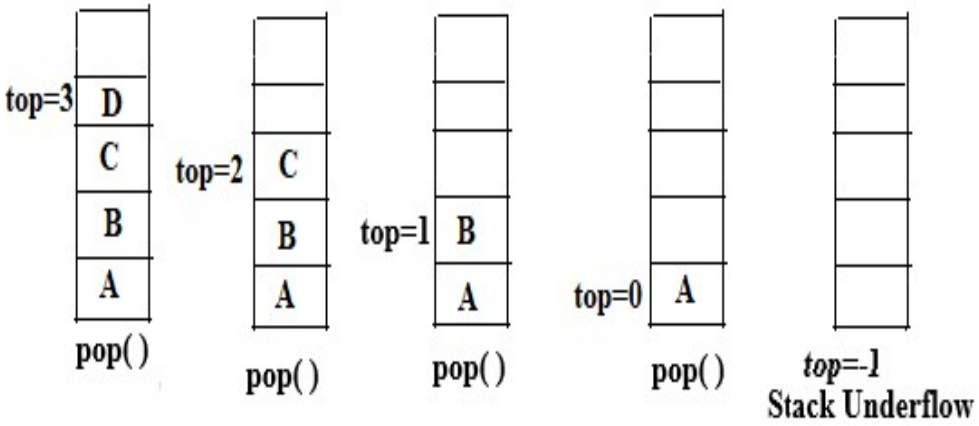
Step 2 : If the stack **is empty**, produces an **Underflow Message** and exit.

Step 3 : If the stack is **not empty**, **accesses the data element at which top** is pointing.

Step 4 : **Decreases the value of top by 1.**

Pop Operation algorithm

```
int pop()
begin
        if(top==-1)
        {
                printf(“Stack is empty.\n");
        }
        else
        {
                ele = stack[top];
                top = top - 1;
                printf(“%d is deleted ” ele);
        }
end
```



PEEK () algorithm

```
Peek()
Begin
        if( top == -1)
        {

printf("stack empty");
        }
        else
        {
                item = stack[top]
                printf("%d",item);
        }
End
```

isfull() algorithm

```
Isfull()
 if (top == MAXSIZE-1)
        write  "stack is full"
```

isempty() algorithm

```
 isempty()
if (top ==-1)
        write "Stack is empty"
```

Stack Operations

Stack Operations

faceprep

```
#include"stdio.h"
#define Maxsize 5
int stack[Maxsize];
int top=-1,ele,choice;
void push();
void pop();
void isfull();
void isempty();
void peek();
void display();
main()
{
        do
        {
         printf("\n*****MENU*****");
        printf( "\n 1.PUSH \t 2.POP \t3.ISFULL
                4.ISEMPTY  \t 5.PEEK \t 6.DISPLAY " );
        printf("\n Enter ur Choice:");
        scanf("%i",&choice);
    switch(choice)
      {
            case 1:printf("\n Enter any Val:");
                   scanf("%d",&ele)
                    push(ele);
                    break;
            case 2:pop();
                   break;
            case 3:isfull();
                   break;
            case 4:isempty();
                   break;
            case 5:peek();
                   break;
            case 6:display();
                   break;
            default :printf("Invalid Choice");
       }
}while(choice<=8);
}
```

```
void push(int ele)
{
        if(top==Maxsize-1)
        {
        printf(“Stack Over flow\n");
        return;
        }
         else
         {
        top = top + 1;
        stack[top] = ele;
          }
}
```

```
void pop()
{
        if(top==-1)
        {
        printf("\n Stack is under folw");
        return;
        }
        item=stack[top];
        top=top-1;
        printf("\n %i is deleted",item);
}
```

```
void isfull()
{
        if(top==Maxsize-1)
            printf("\nStack is Full");
        else
          printf("\nStack is not Full");
}
```

```
void isempty()
{
        if(top==-1)
            printf("\nStack is Empty");
        else
          printf("\nStack is not Empty");
}
```

```
void peek()
{
        if(top==-1)
        {
        printf("\n Stack is Empty ");
        return;
        }
      printf("%i the Top element is:“,stack[top]);
}
```

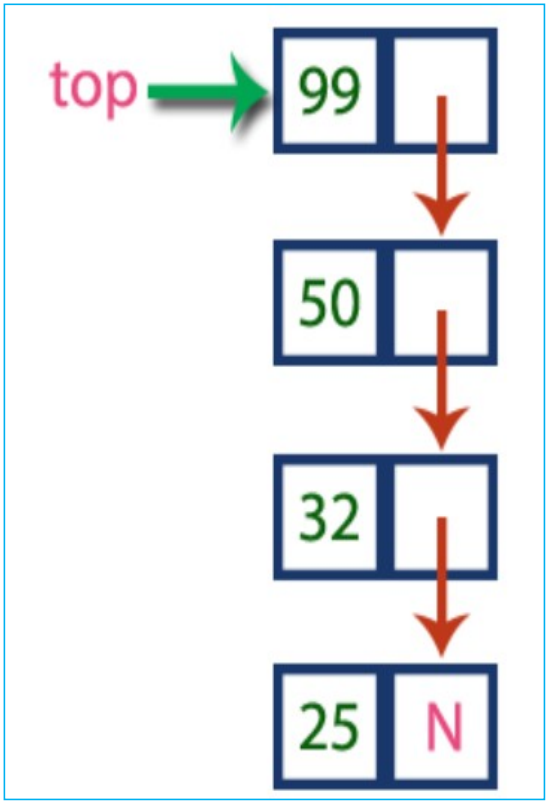
```
void display()
{
        int i;
        if(top==-1)
        {
        printf("\n Stack is empty");
        return;
        }
        for(i=0;i<=top;i++)
        {
        printf("%i",stack[i]);
        }
}
```

Implement of stack Linked list

- The **problem** with the **stack using an array** is, it works only for a **fixed number of data values**. That means the **size of stack must be specified** at the **beginning itself.**
- **Stack using an array** is **not suitable**, **when we don't know the size of data .**
- A **stack can be also** be **implement** by **using a linked list.**
- The **stack using linked list can work** for an **unlimited number of values**. So, there is **no need to fix the size at the beginning**.
- In linked list stack, **every new element** is inserted as **'top' element**.
- Whenever **we want to remove** an **element from the stack, simply** remove the **node** which is pointed by **'top'** and **shift** the 'top' **to its previous node** in the list.
- The **next field** of the **first** node must be **always NULL**.

Implement of stack Linked list

- ✓ To **implement a stack using a linked list**, we **need to set the following things** before implementing actual operations:



Step 1 : **Include all the header files** which are used in the program. And **declare all** the user defined **functions.**

Step 2 : Define a **'Node' structure** with **two members data** and **next.**

Step 3 : Define a **Node pointer 'top'** and **set** it to **NULL.**

Step 4: **Implement** the **main method** by **displaying Menu** with list of operations and make suitable function calls in the main method.

Inserting an element into the Stack

push(ele) :

Step 1 : Create a newNode with given value.

Step 2 : Check whether stack is Empty (top == NULL)

Step 3 : If it is Empty, then set newNode → data = ele and newNode → next = NULL,

Step 4 : If it is Not Empty, then set newNode → data = ele and newNode → next = top.

Step 5 : Finally, set top = newNode.

```
void push(int ele)
{
        struct Node *temp=NULL;
        temp=(struct Node*)malloc(sizeof(struct Node));
        if(top==NULL)
        {
                temp->data=ele;
                temp->next=NULL;
                top=temp;
        }
        else
        {
                temp->data=ele;
                temp->next=top;
                top=temp;
        }
}
```

Deleting an Element from a Stack

Pop() :

Step 1 - Check whether stack is Empty (top == NULL).

Step 2 - If it is Empty, then display "Stack is Empty!!!

Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'(i.e,temp=top).

Step 4 - Then set 'top = top → next'.

Step 5 - Finally, delete 'temp'. (free(temp)).

```
void pop()
{
        if(top==NULL)
        {
                printf("\n Stack is Empty");
                return;
        }
        else
        {
                struct Node *temp=NULL;
                temp=top;
                ele=temp->data;
                top=top->next;
                free(temp);
                printf("%3d is deleted",ele);
        }
}
```

Displaying stack of elements

Display( )

Step 1 : Check whether stack is Empty (top == NULL).

Step 2 : If it is Empty, then display 'Stack is Empty'

Step 3 : If it is Not Empty, then define a Node pointer 'temp' and initialize it with top.

Step 4 : Display 'temp → data --->' and move it to the next node.

Repeat the same until temp reaches to the first node in the stack.

```
void display()
{
        struct Node *temp=NULL;
        temp=top;
        if(top==NULL)
        {
                printf("\n Stack is Empty");
                return;
        }
        else
        {
                while(temp!=NULL)
                {
                printf("%3d",temp->data);
                temp=temp->next;
                }
        }
}
```

```
//Program for stack using linked list
#include"stdlib.h“
#include”stdio.h”
struct Node
{
int data;
struct Node *next;
}*top=NULL;
int choice,ele;
void push(int);
void pop();
void display();

void main()
{

        do
        {
        printf("***Menu***");
        printf("\n 1.Push \t 2.Pop \t 3.Display  \t 4.Peek\n");
        printf("\n Enter your choice");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:printf("\n Enter any val:");scanf("%d",&ele);push(ele);break;
                case 2:pop();break;
                case 3:display();break;
                case 4:peek();break;
        }
        }while(choice<5);
}
```

Stack applications

- Stacks can be **used for expression evaluation.**
- Stacks can be used to **check parenthesis matching in an expression.**
- Stacks can be used for **Conversion from one form of expression to another.**
- Stacks can be used for **Memory Management.**
- Stack data structures are used in **backtracking problems.**

Expressions

- An expression consists of operands and operators.
- C supports 3 types of expressions, they are:

i.Infix Expression	**Ex:**A + B
ii.Prefix Expression	**Ex:**+ A B
iii.Postfix Expression	**Ex:**A B +

- **Operator precedence :**

  It determines which operation is performed first in an expression with more than one operators with different precedence.

Example: 10 + 20 * 30

- **Operators Associativity:**

  It is used when two operators of same precedence appear in an expression. Associativity can be either Left to Right or Right to Left.

Evaluation of expression	Description of each operation
6*2/(2+1 * 2/3 +6) +8 * (8/4)	An expression is given.
6*2/(2+2/3 + 6) + 8 * (8/4)	2 is multiplied by 1, giving value 2.
6*2/(2+0+6) + 8 * (8/4)	2 is divided by 3, giving value 0.
6*2/ 8+ 8 * (8/4)	2 is added to 6, giving value 8.
6*2/8 + 8 * 2	8 is divided by 4, giving value 2.
12/8 +8 * 2	6 is multiplied by 2, giving value 12.
1 + 8 * 2	12 is divided by 8, giving value 1.
1 + 16	8 is multiplied by 2, giving value 16.
17	1 is added to 16, giving value 17.

Conversion of INFIX expression to POSTFIX expression

- **Step 1** : Scan the Infix Expression from **left to right.**
- **Step 2** : If the **scanned character is an operand**, append it **with final Postfix string**.
- **Step 3** : If **the scanned character** is an **'(' or '[' or '{', push it to the stack.**
- **Step 4** : If the **scanned character** is an **')'or ']' or '}', pop the stack and shift it postfix area until a '(' or '[' or '{'** respectively is brace or bracket encountered, and discard both the parenthesis.
- **Step 5** : If the scanned character is an **operator:**

  i. If the **stack is empty** or contains a **left parenthesis** on top, **push** the **operator** on to the **stack.**

  ii. If the **operator** has **higher precedence** than the top of the stack, **push it on the stack**.

Conversion of INFIX expression to POSTFIX expression

iii. If the **operator** has **lower precedence** than the top of the stack, pop **the operator** available on the top of the stack. Then **test the incoming operator** against the **new top of the stack.**

iv. If the **operator** has the **same precedence** with the top of the stack then **use the associativity** rules.

v. If the **associativity** is from **left to right** then **pop the operator** available on the top of the stack then **push the operator**.

vi. If the associativity is from right to left then **push the incoming operator**

- **Step 6** : **Repeat steps 2-6** until infix expression is scanned.
- **Step 7** : Print the output
- **Step 8** : Pop and output from the stack until it is not empty.

Conversion of INFIX expression to POSTFIX expression

OPERATOR	PRECEDENCE	VALUE
Exponentiation ($ or ↑ or ^)	Highest	3
*, /	Next highest	2
+, -	Lowest	1

Example to Convert Infix to Postfix using stack

a + (b*c)

Read character	Stack	Output
a	Empty	a
+	+	a
(	+(	a
b	+(	ab
*	+(*	ab
c	+(*	abc
)	+	abc*
		abc*+

Conversion of INFIX expression to POSTFIX expression

The infix expression is

$(P/(Q-R)*S+T)$

Symbol	Stack	Expression
(	(	—
P	(	P
/	(/	P
(	(/(	P
Q	(/(	PQ
–	(/(–	PQ
R	(/(–	PQR
)	(/	PQR–
*	(*	PQR–/
S	(*	PQR–/S
+	(+	PQR–/S*
T	(+	PQR–/S*T
)		PQR–/ST*+

So, the postfix expression is $PQR-/ST+*$.

Infix Expression: (A/(B-C)*D+E)

Symbol Scanned	Stack	Output
(	(	-
A	(	A
/	(/	A
(	(/(	A
B	(/(	AB
-	(/(-	AB
C	(/(-	ABC
)	(/	ABC-
*	(*	ABC-/
D	(*	ABC-/D
+	(+	ABC-/D*
E	(+	ABC-/D*E
)	Empty	ABC-/D*E+

Postfix Expression: ABC-/D*E+

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack , that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack , that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

Infix Expression:

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

where **^** is an exponential operator.

```
//Program to convert given infix expression into postfix
#include<stdio.h>
#include<ctype.h>
#define max 10
void push(char);
char pop();
int priority(char);
char stack[max];
int top=-1;
int main()
{
char exp[30],x;
int i;
printf("enter the expression\n");
scanf("%s",&exp);
```

```
for(i=0;exp[i]!='\0';i++)
{
        if(isalpha(exp[i]))
                printf("%c",exp[i]);
        else if(exp[i] =='(')
                push(exp[i]);
        else if(exp[i]==')')
        {
                while((x=pop())!='(')
                printf("%c",x);
        }
        else
        {
            while(priority(stack[top])
>=priority(exp[i]))
                {
                printf("%c",pop());
                }
        push(exp[i]);
        }
} // end of the for loop
```

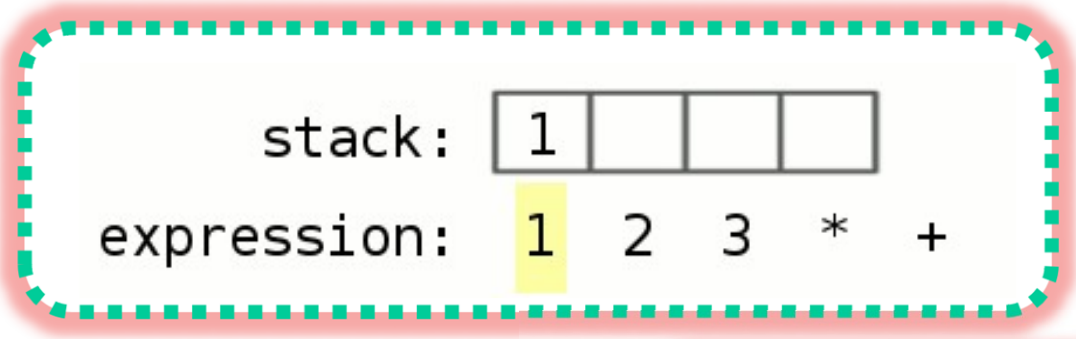
```
while(top!=-1)
{
printf("%c",pop());
}
return 0;
}
```

```
int priority(char x)
{
        if(x=='(')
                return 0;
        else if ((x=='+')||(x=='-'))
                return 1;
        else if((x=='*')||(x=='/')||(x=='%'))
                return 2;
        else if(x=='^')
                return 3;
}
```

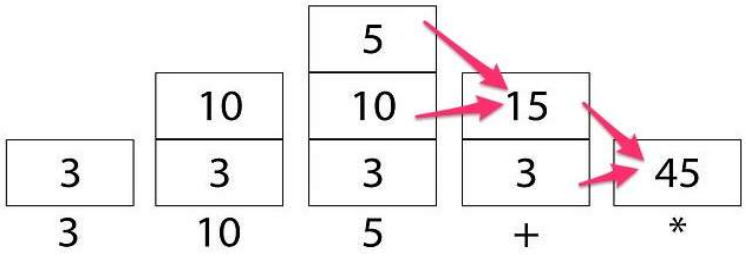
```
void push(char x)
{
        if(top==max-1)
                printf("stack overflow\n");
        else
                stack[++top]=x;
}
char pop()
{
        if(top==-1)
                printf("stack underflow\n");
        else
                return stack[top--];
}
```

Postfix Expression Evaluation

- **Step 1** : Create a **stack to store operands** (or values).
- **Step 2** : **Scan the given expression** and do following **for every scanned character.**
- **Step 3** : **If the scanned character** is a **operand, push it** into the **stack**
- **Step 4**: If the **scanned character** is a **operator**, pop top two operands from stack and push the result back to the stack
- **Step 5** : **Repeat steps 3-4** until the expression is ended, the number in the stack is the final answer



3 10 5 + *



Postfix → 4 3 2 * + 5 -

Operator/Operand	Action	Stack
4	Push	4
3	Push	4, 3
2	Push	4, 3, 2
*	Pop (2, 3) and 3*2 = 6 then Push 6	4, 6
+	Pop (6, 4) and 4+6 = 10 then Push 10	10
5	Push	10, 5
-	Pop (5, 10) and 10-5 = 5 then Push 5	5

Evaluate below postfix expression 234+*6-

Step No.	Value of i	Operation	Stack
1	2	Push 2 in stack	**2**
2	3	Push 3 in stack	**3** **2**
3	4	Push 4 is stack	**4** **3** **2**
4	+	Pop 2 elements from stack and perform addition operation. And push result back to stack. i.e. 4+3 = 7	7 **2**
5	*	Pop 2 elements from stack and perform multiplication operation. And push result back to stack. i.e. 7 * 2 = 14	**14**
6	6	Push 6 in stack	**6** **14**
7	-	Pop 2 elements from stack and perform substraction operation. And push result back to stack. i.e. 14 - 6 = 8	**8**
8		Pop result from stack and display	

```
// program for postfix expr  evaluation
#include<stdio.h>
#include<ctype.h>
#include"string.h"
#define max 10
void push(int);
int pop();
int operation(char,int,int);
int stack[max];
int top=-1,ele,val;
char ch;
int main()
{
char exp[30];
int i,num1,num2,res;
printf("enter the expression\n");
gets(exp);
```

```
for(i=0;exp[i]!='\0';i++)
{
        ch=exp[i];
        if(isalpha(exp[i]))
        {
                printf("\n Enter Value for %c:",exp[i]);
                scanf("%d",&val);
                push(val);
        }
        else
        {
                num1=pop();
                num2=pop();
                val=operation(ch,num1,num2);
                push(val);
        }
}
res=pop();
printf("%d",res);
}
```

```
void push(int ele)
{
        if(top==max-1)
                printf("stack overflow\n");
        else
                stack[++top]=ele;
}


int pop()
{
        if(top==-1)
                printf("stack underflow\n");
        else
                return stack[top--];
}
```

```
int operation(char op,int n1,int n2)
{
        int n3;
        switch(op)
        {
                case '+':n3=n1+n2;break;
                case '-':n3=n1-n2;break;
                case '*':n3=n1*n2;break;
                case '/':n3=n1/n2;break;
                case '%':n3=n1%n2;break;
        }
return n3;
}
```


T
H
A
N
'Q'