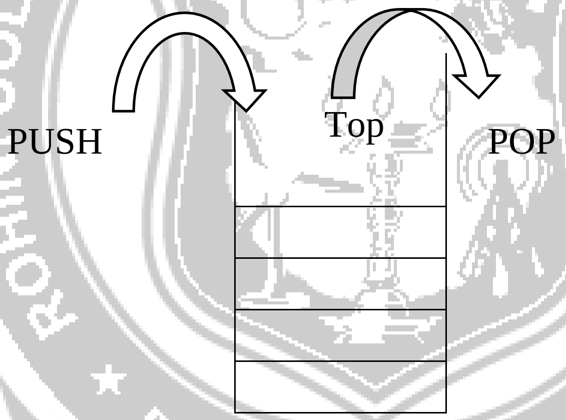
## STACK ADT:

Stack is a linear data structure in which the insertion and deletion operations are performed at only one end.

In a stack, adding and removing of elements are performed at single position which is known as "top". That means, new element is added at top of the stack. In stack, the insertion and deletion operations are performed based on LIFO (Last In First Out) principle.

In a stack, the insertion operation is performed using a function called "push" and deletion operation is performed using a function called "pop".

In the figure, PUSH and POP operations are performed at top position in the stack. That means, both the insertion and deletion operations are performed at one end (i.e., at Top)



### Operations on Stack (Stack ADT)

Two fundamental operations performed on the stack are PUSH and POP.

a) PUSH:

It is the process of inserting a new element at the Top of the stack. For every push operation:

1. Check for Full stack ( overflow ).
2. Increment Top by 1. (Top = Top + 1)
3. Insert the element X in the Top of the stack.

b) POP:

It is the process of deleting the Top element of the stack. For every pop operation:

1. Check for Empty stack ( underflow ).

2. Delete (pop) the Top element X from the stack
3. Decrement the Top by 1. (Top = Top - 1 )

**Exceptional Conditions of stack**

1. **Stack Overflow**

   a) An Attempt to insert an element X when the stack is Full, is said to be stack overflow.
   b) For every Push operation, we need to check this condition.

2. **Stack Underflow:**

   a) An Attempt to delete an element when the stack is empty, is said to be stack underflow.
   b) For every Pop operation, we need to check this condition.

**<u>IMPLEMENTATION OF STACK</u>**

Stack can be implemented in two ways.

1. Static Implementation (Array implementation of Stack)
2. Dynamic Implementation (Linked List Implementation of Stack)

**Array Implementation of Stack**

- Each stack is associated with a Top pointer.
- For Empty stack, Top = -1.
- Stack is declared with its maximum size.

**Array Declaration of Stack:**

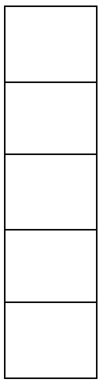
```
#define ArraySize 5int S [ Array Size];
or
int S [ 5 ];
```

**(i) Stack Empty Operation:**

- Initially Stack is Empty.
- With Empty stack Top pointer points to – 1.
- It is necessary to check for Empty Stack before deleting (pop) an element from the stack.

**Routine to check whether stack is empty**

```
int IsEmpty ( Stack S )
{
   if( Top = = - 1 )
   return(1);
}
```



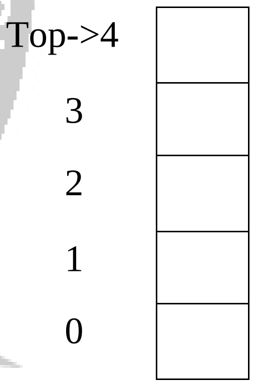
EMPTY Stack
Top= -1

**(ii) Stack Full Operation:**

- As we keep inserting the elements, the Stack gets filled with the elements.
- Hence it is necessary to check whether the stack is full or not before inserting a new element into the stack.

**Routine to check whether a stack is full**

```
int IsFull ( Stack S )

    {   if( Top = = Arraysize – 1 )

        return(1);

    }
```



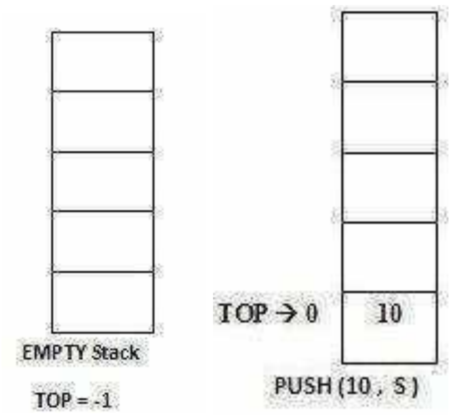
**FULL Stack**

**(ii) Push Operation**

1. It is the process of inserting a new element at the Top of the stack.
2. It takes two parameters. Push(X, S) the element X to be inserted at the Top of the StackS.
3. Before inserting an Element into the stack, check for Full Stack.
4. If the Stack is already Full, Insertion is not possible.
5. Otherwise, Increment the Top pointer by 1 and then insert the element X at the Top of theStack.

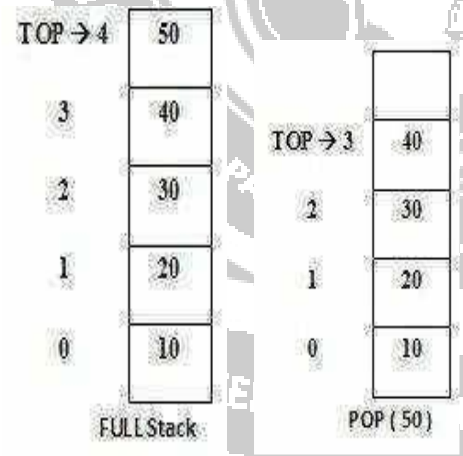
*Routine to push an element into the stack*

```
    void Push ( int X , Stack S )
    {
     if ( Top = = Arraysize - 1)
           Error("Stack is full!!Insertion is not possible");
      else
          {
          Top = Top +1;
          S [ Top ] =X;
        }
    }
```



**(iii) Pop Operation**

- It is the process of deleting the Top element of the stack.
- It takes only one parameter. Pop(X).The element X to be deleted from the Top of theStack.
- Before deleting the Top element of the stack, check for Empty Stack.
- If the Stack is Empty, deletion is not possible.
- Otherwise, delete the Top element from the Stack and then decrement the Top pointer by1



**Routine to Pop the Top element of the stack**

```
void Pop ( Stack S )
{
if ( Top = = - 1)
Error ( "Empty stack! Deletion not possible");
else
{
X = S [ Top ] ;
```

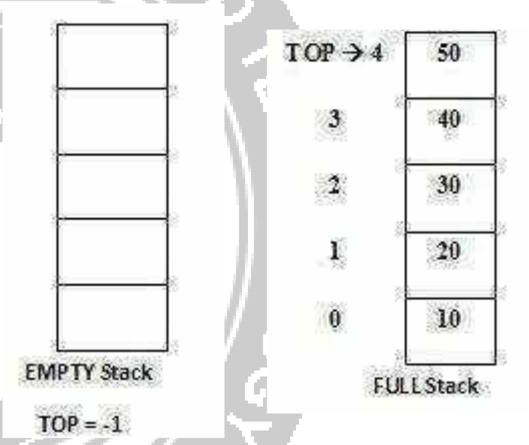
Top = Top – 1;
}
}

**Return Top Element**

- Pop routine deletes the Top element in the stack.
- If the user needs to know the last element inserted into the stack, then the user can returnthe Top element of the stack.
- To do this, first check for Empty Stack.
- If the stack is empty, then there is no element in the stack.
- Otherwise, return the element which is pointed by the Top pointer in the Stack.

**Routine to return top Element of the stack**

```
int TopElement(Stack S)
{
  if(Top==-1)
  {
     Error(“Empty stack!!No elements”);
     return 0;
  }
 else
     return S[Top];
}
```



**Linked list implementation of Stack**

- Stack elements are implemented using SLL (Singly Linked List) concept.
- Dynamically, memory is allocated to each element of the stack as a node.

**Type Declarations for Stack using SLL**

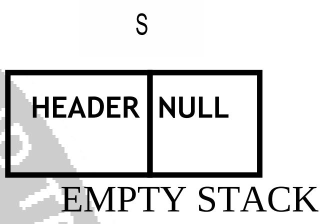
```
struct node;
typedef struct node *stack;
typedef struct node *position;
stack S;
struct node
{
intdata;
position next;
};
int IsEmpty(Stack S);
void Push(int x, Stack S);
void Pop(Stack S);
int TopElement(Stack S);
```

**(i) Stack Empty Operation:**

- Initially Stack is Empty.
- With Linked List implementation, Empty stack is represented as S -> next = NULL.
- It is necessary to check for Empty Stack before deleting (pop) an element from the stack.

**Routine to check whether the stack is empty**

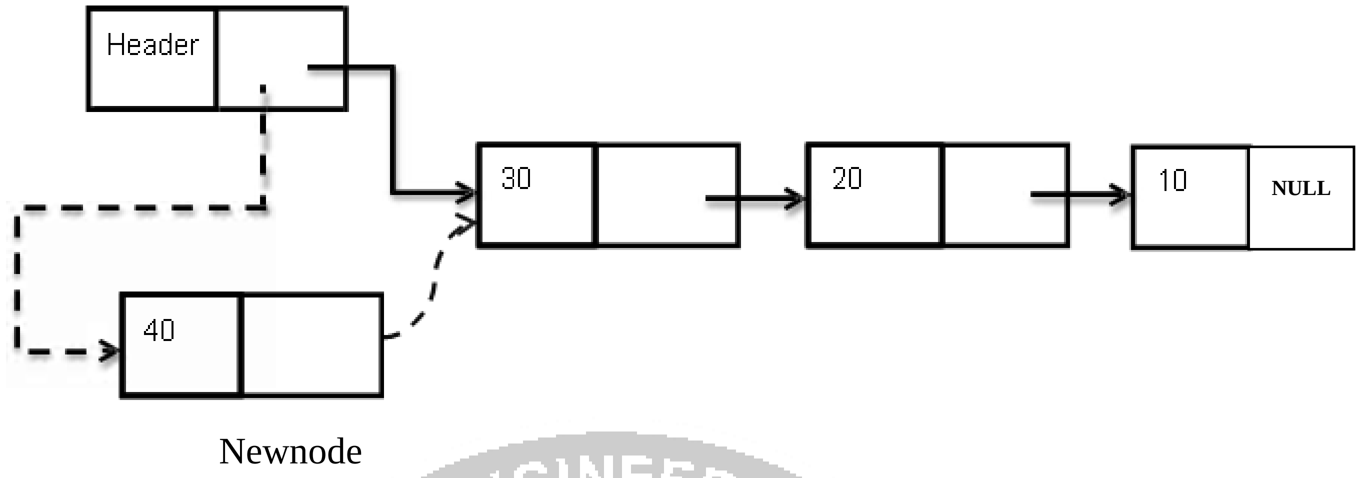
```
int IsEmpty( Stack S)
{
if ( S -> next = = NULL)
   return ( 1 );
}
```



**(ii) Push Operation**

- It is the process of inserting a new element at the Top of the stack.
- With Linked List implementation, a new element is always inserted at the Front of theList. (i.e.) S -> next.
- It takes two parameters. Push(X, S) the element X to be inserted at the Top of the Stack S.
- Allocate the memory for the newnode to be inserted.
- Insert the element in the data field of the newnode.
- Update the next field of the newnode with the address of the next node which is stored in the S -> next.

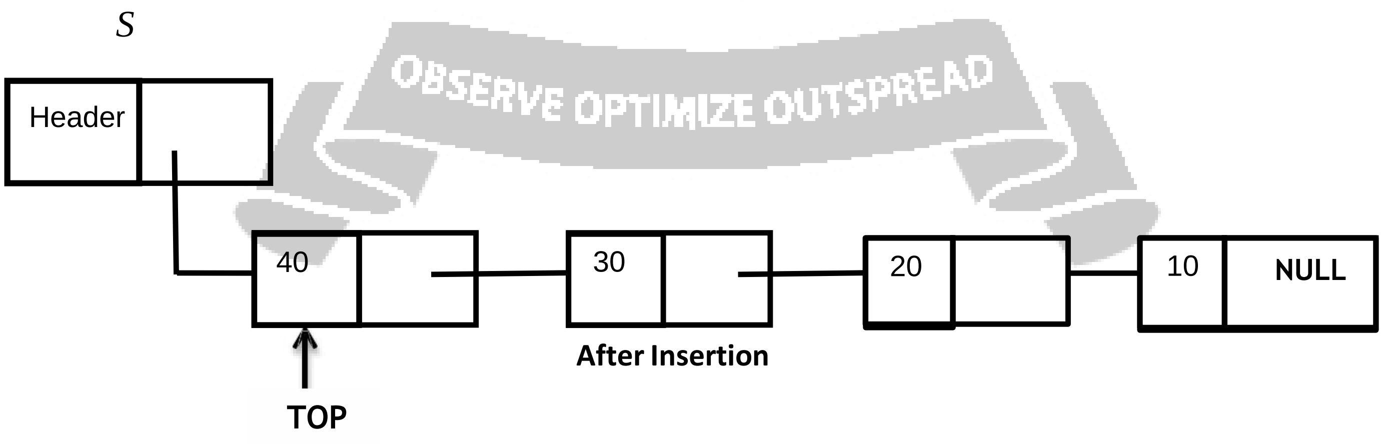
S



**Before Insertion**

**Push routine** /*Inserts element at front of the list

```
void push(int X, Stack S)
{
        Position newnode, Top;
            newnode = malloc (sizeof( struct node ) );
            newnode -> data = X;
            newnode -> next = S -> next;
            S -> next = newnode;
            Top = newnode;
}
```

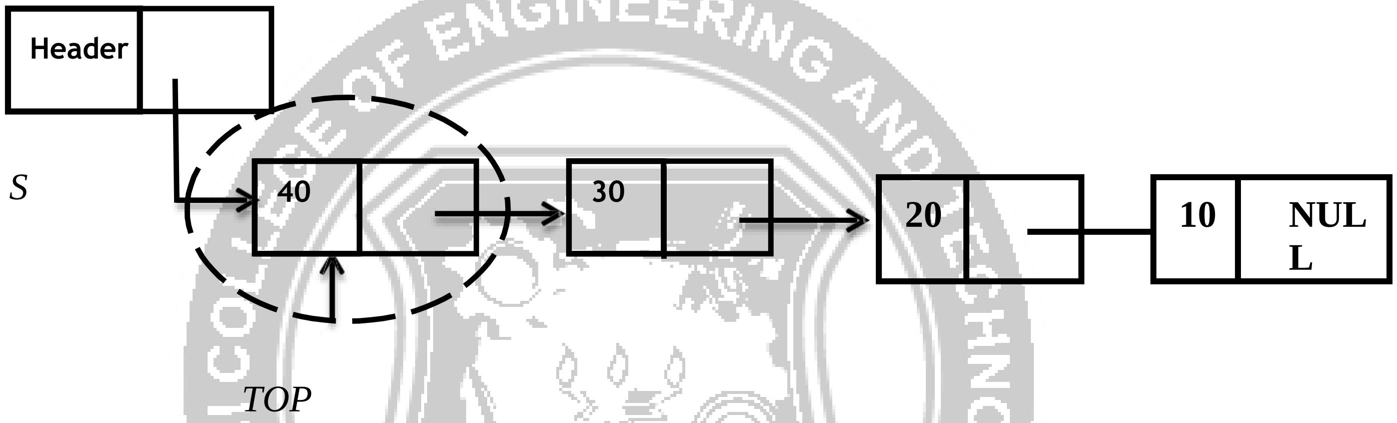


**After Insertion**

**(iii) Pop Operation**

- It is the process of deleting the Top element of the stack.
- With Linked List implementations, the element at the Front of the List(i.e.) S -> next is always deleted.

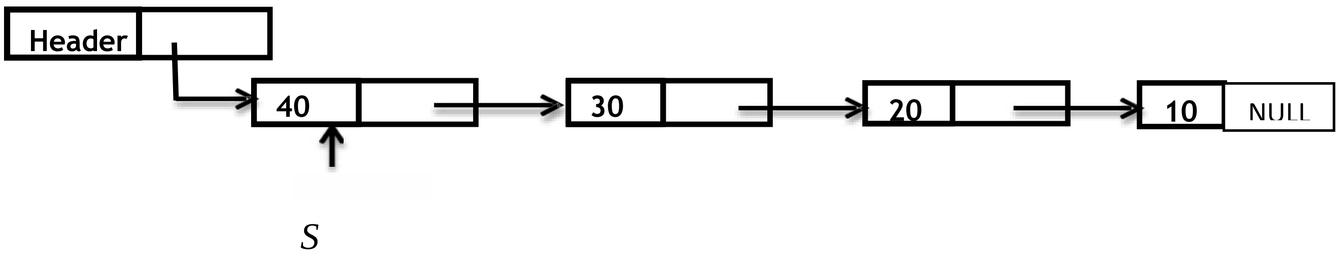
- It takes only one parameter. Pop(X).The element X to be deleted from the Front of theList.
- Before deleting the front element in the list, check for Empty Stack.
- If the Stack is Empty, deletion is not possible.
- Otherwise, make the front element in the list as "temp".
- Update the next field of header.
- Using free ( ) function, Deallocate the memory allocated for temp node

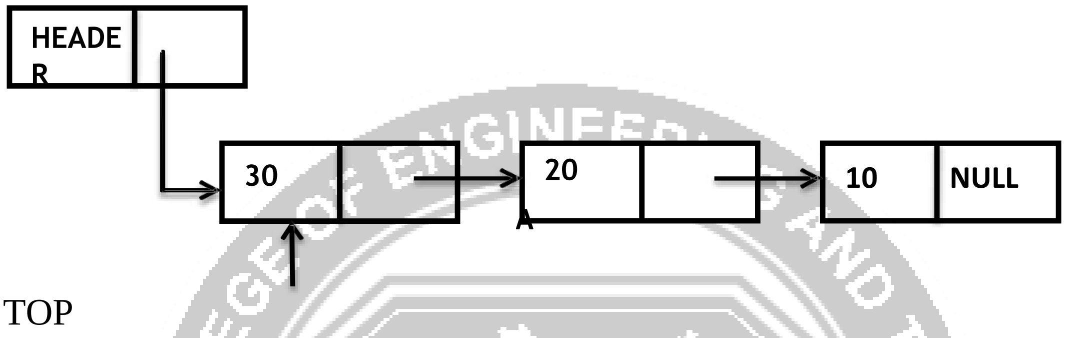


Before Deletion

**Pop routine** **/*Deletes the element at front of list**

```
void Pop( Stack S )
{
          Position temp, Top;
           Top = S -> next;
           if( S -> next = = NULL)
                    Error("empty stack! Pop not possible");
           else
          {
                    Temp = S -> next;
                    S -> next = temp -> next;
                    free(temp);
                    Top = S -> next;
          } }
```



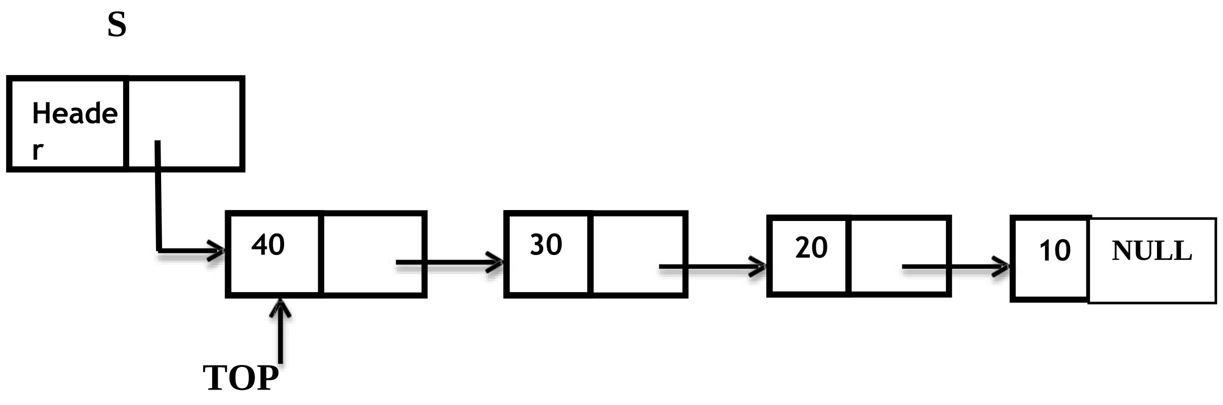


**(iv) Return Top Element**

- Pop routine deletes the Front element in the List.
- If the user needs to know the last element inserted into the stack, then the user can return the Top element of the stack.
- To do this, first check for Empty Stack.
- If the stack is empty, then there is no element in the stack.
- Otherwise, return the element present in the S -> next -> data in the List.

**Routine to Return Top Element**

```
int TopElement(Stack S)

{

   if(S->next==NULL)

     {

         error("Stack is

         empty");return 0;

   else

         return  S->next->data;

}
```


S
Heade
r
40
30
20
10
NULL
TOP