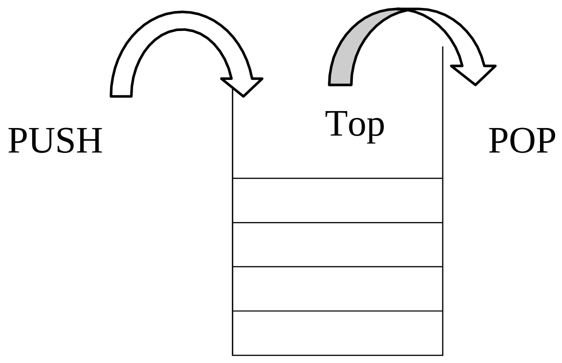
STACK

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

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 pushoperation:

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.

Implementation of Stack

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:

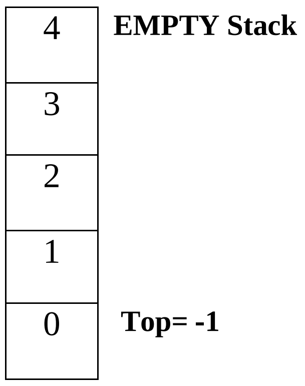
```
#define SZ 5

int S [ SZ]
```

(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);
}
```

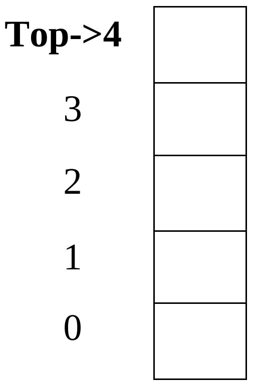


ii) Stack Full Operation:

- 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);

    }
```



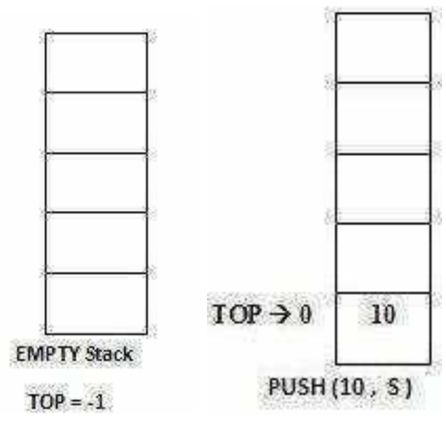
FULL Stack

(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 Stack S.
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 the Stack.

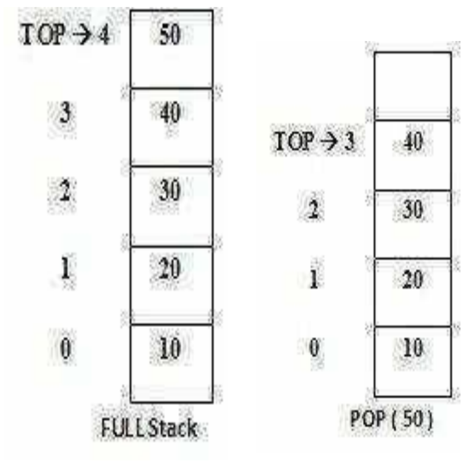
Routine to push an element into the stack

```
void Push ( int X , Stack S )
{
 if ( Top = = SZ - 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 the Stack.
- 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

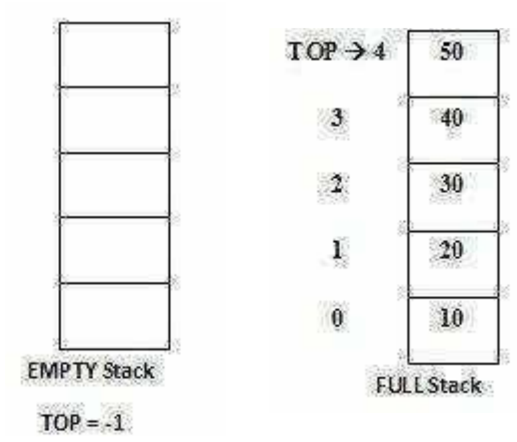
```
void Pop ( Stack S )
{
if ( Top = = - 1)
Error ( "Empty stack! Deletion not possible");
else
{
X = S [ Top ] ;
Top = Top – 1;
}
}
```

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 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 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];
}
```



Stack Program using Static Arrays

```
#include<stdio.h>
#define sz 5
        void main()
                {
                int s[sz], top = -1, item, ch;
                clrscr();

                for(;;)
                {
                printf(" Enter 1:Insert 2: delete 3: Display 4:exit \n");
                scanf("%d",&ch);
                        switch(ch)
                                {
                                case 1: printf("Enter the Item to be Inserted \n");
                                        scanf("%d",&item);
                                        push(s,&top,item);
                                        break;
                                case 2: pop(s,&top);
                                        break;
                                case 3: display(s,top);
                                        break;
                                default: exit(0);
                                }
                }
                }
```

// Function to Push
```
        push(int s[sz], int *top, int item)
        {
        if (*top == sz-1)
        printf(" Stack Overflow, Insertion not Possible \n");
        else
        {
        *top = *top + 1;
        s[*top] = item;
        }
        }
```

// Function to Pop
```
        pop(int s[sz], int *top)
                {
                if (*top == -1)
                printf(" Stack Underflow, deletion not Possible \n");
                else
                {
                printf(" Item deleted is %d \n", s[*top]);
                *top = *top - 1;
                }
                }
```

// Function to Display
```
display(int s[sz], int top)
{
int i;

if (top == -1)
printf(" Stack is Empty, Display not
Possible \n");
else
{
printf(" The content of Stack are \n");
for(i=0;i<=top;i++)
printf("s[%d] ===>> %d\n",i,s[i]);
}
`
```

Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)
a. Push an Element on to Stack b. Pop an Element from Stack
c. Demonstrate Overflow and Underflow situations on Stack
d. Display the status of Stack f. Exit.
Support the program with appropriate functions for each of the above operations

```
#include<stdio.h>
#include<stdlib.h>
int *stack, top=-1,sz=5;
```

// Function to insert elements to stack
```
void push(int ele)
{
   if(top==sz-1)
   {
   printf("Stack overflow\n");
   sz++;
   printf("Stack size increased to %d\n",sz);
   stack=(int*)realloc(stack,sz*sizeof(int));
    }
    top++;
    stack[top] = ele;
    printf("%d pushed \n",ele);

 }
```

// Function to Delete item from stack
```
int pop()
{
   int ele;
   if(top==-1)
   {
   printf("Stack underflow\n");
   exit(0);
   }
   else
   {
   ele=stack[top];
    printf("popped item is %d\n",ele);
    --top;
    sz--;
    printf("Stack capacity decreased to %d \n",sz);
    stack=(int *)realloc(stack,sz*sizeof(int));
    }
```

```
// Main Program
void main()
{
   int ch,item;
   stack=(int *)malloc(sz*sizeof(int));
   clrscr();
   for(;;)
   {
          printf("Enter your choice 1.push 2.pop 3.Quit\n");
          scanf("%d",&ch);

          switch(ch)
          {
             case 1:printf("Enter the element to push :\n");
             scanf("%d",&item);
             push(item);
             break;
             case 2:pop();
             break;
             case 3: exit(0);
     }
   }
}
```