Stack

A stack is a basic data structure, where insertion and deletion of items takes place at one end called top of the stack. This structure is used all throughout programming. The basic concept can be illustrated by thinking of your data as a stack of plates or books where you can only take the top item off the stack in order to remove things from it.

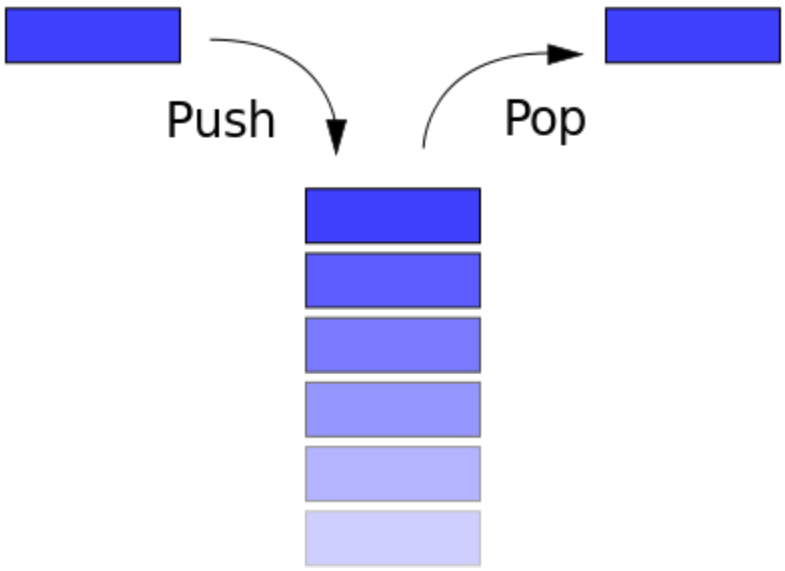
A stack is also called a LIFO (Last In First Out) to demonstrate the way it accesses data. There are basically three operations that can be performed on stacks . They are

1) inserting an item into a stack (push).

2) deleting an item from the stack (pop).

3) displaying the contents of the stack(peek).

\



CODES (C)

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<string.h>
typedef struct stack
{
	int *a;
	int tos,ms;
}st;

void init(st *s,int n)
{
	s->ms=n;
	s->a=(int *)malloc(s->ms*sizeof(int));
	s->tos=-1;
}

int isfull(st *s)
{
	if(s->tos==s->ms-1)
		return 1;
	else
		return 0;
}

int isempty(st *s)
{
	if(s->tos==-1)
		return 1;
	else
		return 0;
}

void push(st *s,int z)
{
	if(isfull(s)==1)
		printf("Stack is full");
	else
	{
		s->a[++s->tos]=z;
	}
}
```

```
int pop(st *s)
{
	int a=0;
	if(isempty(s)==1)
		printf("Stack is empty");
	else
	{
		a=s->a[s->tos];
		s->tos--;
	}
	return a;
}
char peek(st *s)
{
	char p;
	if(isempty(s)==1)
		printf("Stack is empty");
	else
	{
		p=s->a[s->tos];
	}
	return p;
}
```

CODES (JAVA)

```
public class Stack
{
   private int max;
   private long[] stackArray;
   private int top;
   public MyStack(int s)
   {
      max = s;
      stackArray = new long[max];
      top = -1;
   }
   public void push(long j)
   {
        if(isFull())
                System.out.println(“Stack overflow”);
        else
                stackArray[++top] = j;
   }
   public long pop() {
      if(isEmpty())
        {
                System.out.println(“Stack underflow”);
                return -999;
        }
        else
                return stackArray[top--] = j;

   }
   public long peek() {
      return stackArray[top];
   }
   public boolean isEmpty() {
      return (top == -1);
   }
   public boolean isFull() {
      return (top == max - 1);
   }
}
```