**Data Structure**

Section #2 "Stack"

 **Stack** is a linear data structure which follows a particular order in which the operations are performed. The order may be **LIFO** (Last In First Out) or **FILO**(First In Last Out).

**Time Complexities of operations on stack:**

push(), pop(), isEmpty() and peek() all take O(1) time. We do not run any loop in any of these operations.

**Applications of stack:**

* Redo-undo features at many places like editors, photoshop.
* Forward and backward feature in web browser
* Used in many algorithms like Tower of Hanoi, [tree traversals](https://www.geeksforgeeks.org/618/), stock span problem, histogram problem.

**Basic Operations of Stack**

A stack is an object or more specifically an abstract data structure(ADT) that allows the following operations:

* **Push**: Add an element to the top of a stack
* **Pop**: Remove an element from the top of a stack
* **IsEmpty**: Check if the stack is empty
* **IsFull**: Check if the stack is full
* **Peek**: Get the value of the top element without removing it

**Implementation:**   
There are two ways to implement a stack:

* **Using array**
* Using linked list

**Write a program to insert n numbers into a stack and print them**

/\* Java program to implement basic stack

operations \*/

class Stack {

static final int MAX = 1000;

int top;

int a[] = new int[MAX]; // Maximum size of Stack

boolean isEmpty()

{

return (top < 0);

}

Stack()

{

top = -1;

}

boolean push(int x)

{

if (top >= (MAX - 1)) {

System.out.println("Stack Overflow");

return false;

}

else {

a[++top] = x;

System.out.println(x + " pushed into stack");

return true;

}

}

int pop()

{

if (top < 0) {

System.out.println("Stack Underflow");

return 0;

}

else {

int x = a[top--];

return x;

}

}

int peek()

{

if (top < 0) {

System.out.println("Stack Underflow");

return 0;

}

else {

int x = a[top];

return x;

}

}

}

// Driver code

class Main {

public static void main(String args[])

{

Stack s = new Stack();

s.push(10);

s.push(20);

s.push(30);

System.out.println(s.pop() + " Popped from stack");

}

}

**Write a program to find the maximum number in stack and print it out**

The same first program except modification of pop function as following:

int pop()

{

int max = a[top--];

for (;;)

{

if (top < 0)

break;

else

{

if (max < a[top])

max = a[top];

top--;

}

}

return max;

}

**Task #2**

**Write a program to find the Minimum number in stack and print it out.**