Stack Data Structure (Introduction and Program)

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).

Mainly the following three basic operations are performed in the stack:

* **Push:**Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
* **Peek or Top:** Returns top element of stack.
* **isEmpty:**Returns true if stack is empty, else false.

[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2013/03/stack.png)

**How to understand a stack practically?**  
There are many real-life examples of a stack. Consider the simple example of plates stacked over one another in a canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO/FILO order.

**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:**

* [Balancing of symbols](https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/)
* [Infix to Postfix](http://quiz.geeksforgeeks.org/stack-set-2-infix-to-postfix/) /Prefix conversion
* Redo-undo features at many places like editors, photoshop.
* Forward and backward feature in web browsers
* Used in many algorithms like [Tower of Hanoi,](https://www.geeksforgeeks.org/recursive-functions/)[tree traversals](https://www.geeksforgeeks.org/618/), [stock span problem](https://www.geeksforgeeks.org/the-stock-span-problem/), [histogram problem](https://www.geeksforgeeks.org/largest-rectangular-area-in-a-histogram-set-1/).
* Other applications can be Backtracking, [Knight tour problem](https://www.geeksforgeeks.org/backtracking-set-1-the-knights-tour-problem/), [rat in a maze](https://www.geeksforgeeks.org/backttracking-set-2-rat-in-a-maze/), [N queen problem](https://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/) and [sudoku solver](https://www.geeksforgeeks.org/backtracking-set-7-suduku/)
* In Graph Algorithms like [Topological Sorting](https://www.geeksforgeeks.org/topological-sorting/) and [Strongly Connected Components](https://www.geeksforgeeks.org/strongly-connected-components/)

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

* Using array
* Using linked list

**Implementing Stack using Arrays**

/\* 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");

    }

}

10 pushed into stack

20 pushed into stack

30 pushed into stack

30 popped from stack

|  |
| --- |
| / Java Code for Linked List Implementation    public class StackAsLinkedList {        StackNode root;        static class StackNode {          int data;          StackNode next;            StackNode(int data)          {              this.data = data;          }      }        public boolean isEmpty()      {          if (root == null) {              return true;          }          else              return false;      }        public void push(int data)      {          StackNode newNode = new StackNode(data);            if (root == null) {              root = newNode;          }          else {              StackNode temp = root;              root = newNode;              newNode.next = temp;          }          System.out.println(data + " pushed to stack");      }        public int pop()      {          int popped = Integer.MIN\_VALUE;          if (root == null) {              System.out.println("Stack is Empty");          }          else {              popped = root.data;              root = root.next;          }          return popped;      }        public int peek()      {          if (root == null) {              System.out.println("Stack is empty");              return Integer.MIN\_VALUE;          }          else {              return root.data;          }      }        public static void main(String[] args)      {            StackAsLinkedList sll = new StackAsLinkedList();            sll.push(10);          sll.push(20);          sll.push(30);            System.out.println(sll.pop() + " popped from stack");            System.out.println("Top element is " + sll.peek());      }  } |

Output:

10 pushed to stack

20 pushed to stack

30 pushed to stack

30 popped from stack

Top element is 20

**Pros:** The linked list implementation of stack can grow and shrink according to the needs at runtime.  
**Cons:** Requires extra memory due to involvement of pointers.