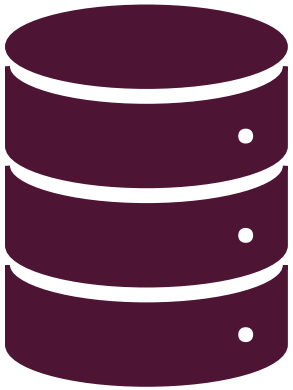


DATA STRUCTURES

WENYE LI
CUHK-SZ

OUTLINE

- Stack and Implementation
- Queue and Implementation
- Examples

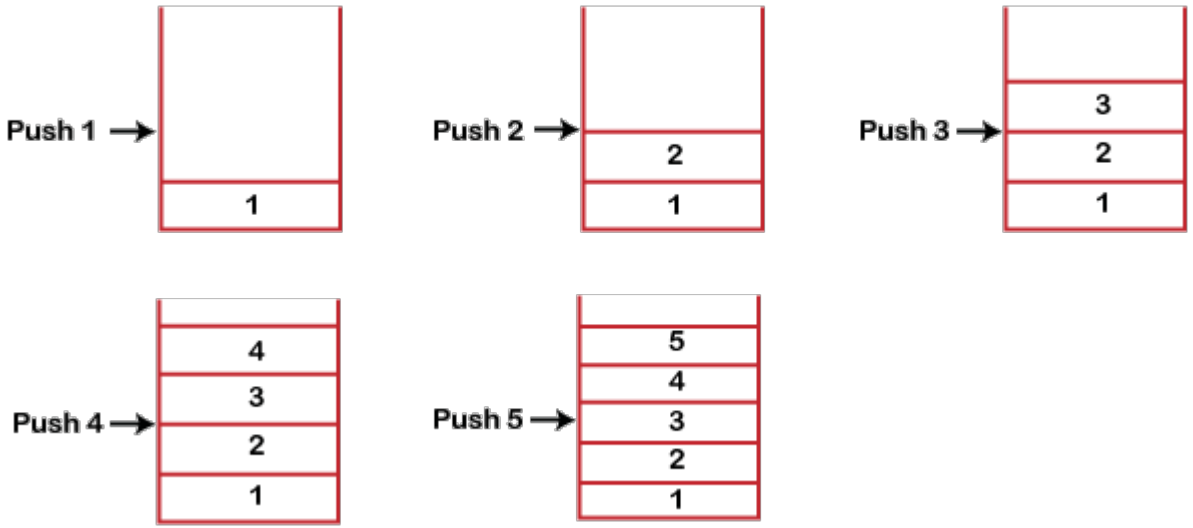
STACK

- Stack as a data structure is not related to the stack memory area!
  - They are completely different things.
  - Refer to stack as a data structure by Stack, and the stack memory area by Stack memory.

- Key points:
  - It is called as stack because it behaves like a real-world stack, piles of books, etc.
  - A Stack is an abstract data type with a pre-defined capacity, which means that it can store the elements of a limited size.
  - It is a data structure that follows some order to insert and delete the elements, and that order can be LIFO (Last In, First Out) or FILO (First In, Last Out).

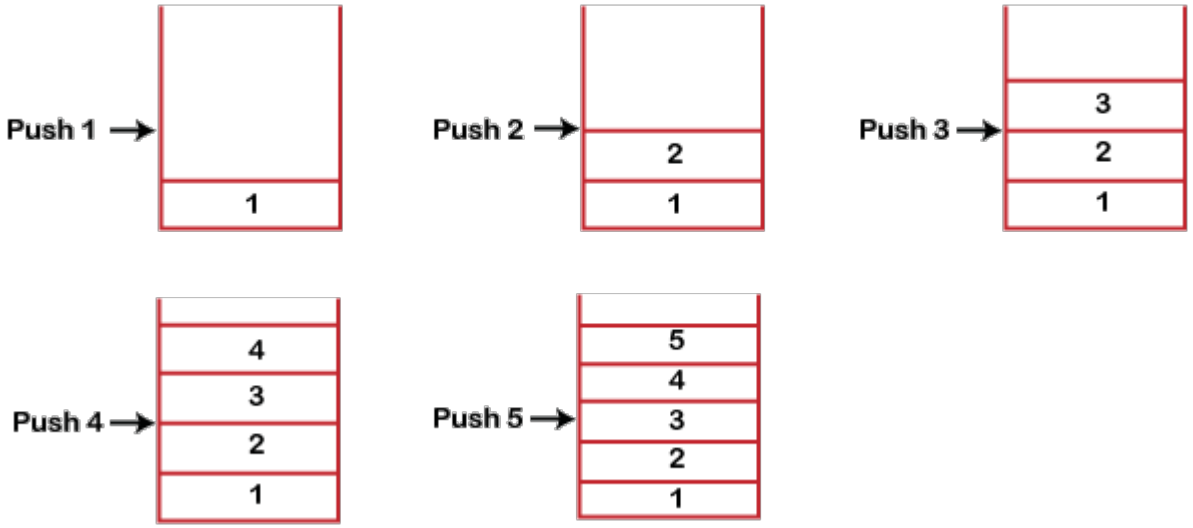
STACK

- Stack works on the LIFO pattern.
  - In the below figure there are five memory blocks in the stack; the size of the stack is 5.
- Suppose we want to store the elements in a stack and let's assume that stack is empty. We are pushing the elements one by one until the stack becomes full.



STACK

- It goes from the top to the bottom when we were entering the new element in the stack. The stack gets filled up from the bottom to the top.
- For the delete operation, it follows the LIFO pattern. The value 1 is entered first, so it will be removed only after the deletion of all the other elements.



STACK

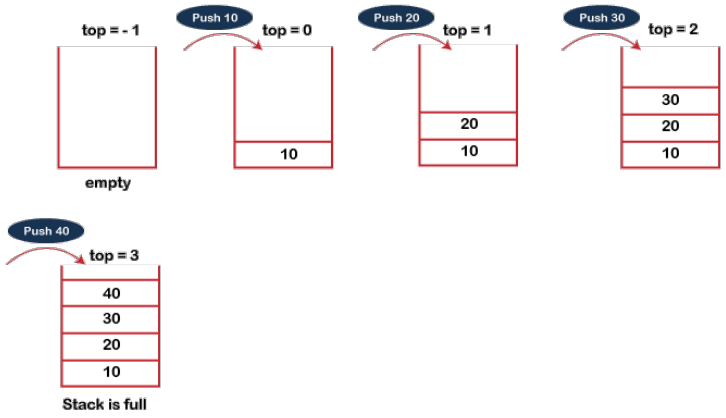
- A Stack is a linear data structure that follows LIFO principle.
- Stack has one end. It contains only one pointer top pointer pointing to the topmost element of the stack.
- Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack.

- Therefore, a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.

OPERATIONS ON STACK

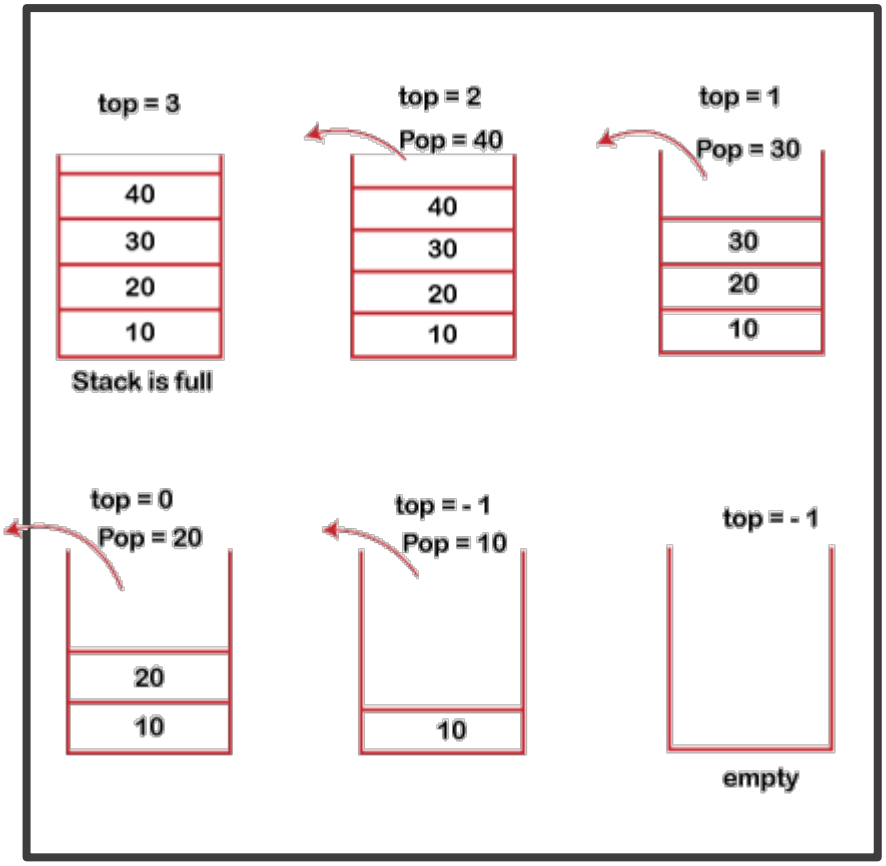
- push(): When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- pop(): When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- isEmpty(): It determines whether the stack is empty or not.
- isFull(): It determines whether the stack is full or not.'
- peek(): It returns the element at the given position.
- count(): It returns the total number of elements available in a stack.
- change(): It changes the element at the given position.
- display(): It prints all the elements available in the stack.

PUSH OPERATION



- Before inserting an element in a stack, we check whether the stack is full.
- If we try to insert the element in a stack, and the stack is full, then the overflow condition occurs.
- When we initialize a stack, we set the value of top as -1 to check that the stack is empty.
- When the new element is pushed in a stack, first, the value of the top gets incremented, i.e., top=top+1, and the element will be placed at the new position of the top.
- The elements will be inserted until we reach the max size of the stack.

POP OPERATION



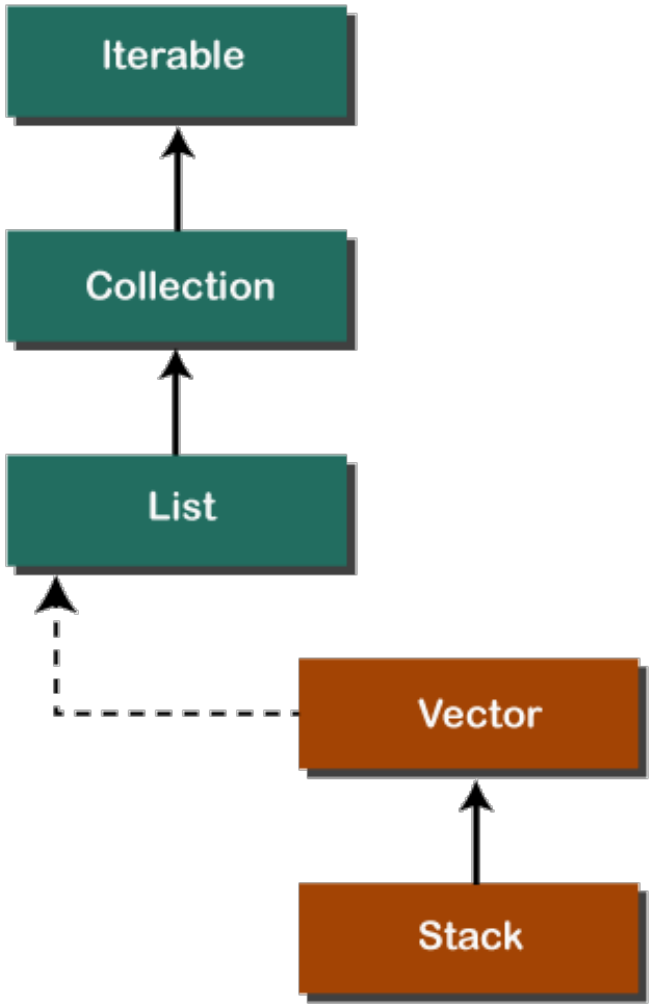
- Before deleting the element from the stack, we check whether the stack is empty.
- If we try to delete the element from the empty stack, then the underflow condition occurs.
- If the stack is not empty, we first access the element which is pointed by the top.
- Once pop operation is performed, the top is decremented by 1, i.e., top=top-1.

OPERATIONS ON STACK

Method	Modifier and Type	Method Description
empty()	boolean	The method checks the stack is empty or not.
push(E item)	E	The method pushes (insert) an element onto the top of the stack.
pop()	E	The method removes an element from the top of the stack and returns the same element as the value of that function.
peek()	E	The method looks at the top element of the stack without removing it.
search(Object o)	int	The method searches the specified object and returns the position of the object.

STACK IN JAVA

- In Java, Stack is a class that falls under the Collection framework that extends the Vector class.
- It also implements interfaces List, Collection, Iterable, Cloneable, Serializable.
- It represents the LIFO stack of objects.
- Before using the Stack class, we must import the java.util package.



```java
import java.util.Stack;

public class StackEmptyMethodExample
{
    public static void main(String[] args)
    {
        //creating an instance of Stack class
        Stack<Integer> stk = new Stack<>();
        // checking stack is empty or not
        boolean result = stk.empty();
        System.out.println("Is the stack empty? " + result);
        // pushing elements into stack
        stk.push(78);
        stk.push(113);
        stk.push(90);
        stk.push(120);
        //prints elements of the stack
        System.out.println("Elements in Stack: " + stk);
        result = stk.empty();
        System.out.println("Is the stack empty? " + result);
    }
}
```

Output:

```
Is the stack empty? true
Elements in Stack: [78, 113, 90, 120]
Is the stack empty? false
```

```java
import java.util.*;
public class StackPushPopExample {
    public static void main(String args[]) {
        //creating an object of Stack class
        Stack <Integer> stk = new Stack<>();
        System.out.println("stack: " + stk);
        //pushing elements into the stack
        pushelmnt(stk, 20);
        pushelmnt(stk, 13);
        pushelmnt(stk, 89);
        pushelmnt(stk, 90);
        pushelmnt(stk, 11);
        pushelmnt(stk, 45);
        pushelmnt(stk, 18);
        //popping elements from the stack
        popelmnt(stk);
        popelmnt(stk);
        //throws exception if the stack is empty
        try {
            popelmnt(stk);
        } catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }
    //performing push operation
    static void pushelmnt(Stack stk, int x) {
        //invoking push() method
        stk.push(new Integer(x));
        System.out.println("push -> " + x);
        //prints modified stack
        System.out.println("stack: " + stk);
    }
    //performing pop operation
    static void popelmnt(Stack stk) {
        System.out.print("pop -> ");
        //invoking pop() method
        Integer x = (Integer) stk.pop();
        System.out.println(x);
        //prints modified stack
        System.out.println("stack: " + stk);
    }
}
```

Output:

```
stack: []
push -> 20
stack: [20]
push -> 13
stack: [20, 13]
push -> 89
stack: [20, 13, 89]
push -> 90
stack: [20, 13, 89, 90]
push -> 11
stack: [20, 13, 89, 90, 11]
push -> 45
stack: [20, 13, 89, 90, 11, 45]
push -> 18
stack: [20, 13, 89, 90, 11, 45, 18]
pop -> 18
stack: [20, 13, 89, 90, 11, 45]
pop -> 45
stack: [20, 13, 89, 90, 11]
pop -> 11
stack: [20, 13, 89, 90]
```

```java
import java.util.Stack;

public class StackPeekMethodExample
{
    public static void main(String[] args)
    {
        Stack<String> stk = new Stack<>();
        // pushing elements into Stack
        stk.push("Apple");
        stk.push("Grapes");
        stk.push("Mango");
        stk.push("Orange");
        System.out.println("Stack: " + stk);
        // Access element from the top of the stack
        String fruits = stk.peek();
        //prints stack
        System.out.println("Element at top: " + fruits);
    }
}
```

Output:

```
Stack: [Apple, Grapes, Mango, Orange]
Element at the top of the stack: Orange
```

```java
import java.util.Stack;
public class StackSearchMethodExample
{
    public static void main(String[] args)
    {
        Stack<String> stk = new Stack<>();
        //pushing elements into Stack
        stk.push("Mac Book");
        stk.push("HP");
        stk.push("DELL");
        stk.push("Asus");
        System.out.println("Stack: " + stk);
        // Search an element
        int location = stk.search("HP");
        System.out.println("Location of Dell: " + location);
    }
}
```

```java
import java.util.Stack;

public class StackSizeExample
{
    public static void main (String[] args)
    {
        Stack stk = new Stack();
        stk.push(22);
        stk.push(33);
        stk.push(44);
        stk.push(55);
        stk.push(66);
        // Checks the Stack is empty or not
        boolean rslt = stk.empty();
        System.out.println("Is the stack empty or not? " + rslt);
        // Find the size of the Stack
        int x = stk.size();
        System.out.println("The stack size is: " + x);
    }
}
```

Output:

```
Is the stack empty or not? false
The stack size is: 5
```

```java
import java.util.Iterator;
import java.util.Stack;
public class StackIterationExample1
{
    public static void main (String[] args)
    {
        //creating an object of Stack class
        Stack stk = new Stack();
        //pushing elements into stack
        stk.push("BMW");
        stk.push("Audi");
        stk.push("Ferrari");
        stk.push("Bugatti");
        stk.push("Jaguar");
        //iteration over the stack
        Iterator iterator = stk.iterator();
        while(iterator.hasNext())
        {
            Object values = iterator.next();
            System.out.println(values);
        }
    }
}
```

Output:

```
BMW
Audi
Ferrari
Bugatti
Jaguar
```

```java
import java.util.*;
public class StackIterationExample2
{
    public static void main (String[] args)
    {
        //creating an instance of Stack class
        Stack <Integer> stk = new Stack<>();
        //pushing elements into stack
        stk.push(119);
        stk.push(203);
        stk.push(988);
        System.out.println("Iteration over the stack using forEach() Method:");
        //invoking forEach() method for iteration over the stack
        stk.forEach(n ->
        {
            System.out.println(n);
        });
    }
}
```

Output:

```
Iteration over the stack using forEach() Method:
119
203
988
```

```java
import java.util.Iterator;
import java.util.ListIterator;
import java.util.Stack;

public class StackIterationExample3
{
    public static void main (String[] args)
    {
        Stack <Integer> stk = new Stack<>();
        stk.push(119);
        stk.push(203);
        stk.push(988);
        ListIterator<Integer> ListIterator = stk.listIterator(stk.size());
        System.out.println("Iteration over the Stack from top to bottom:");
        while (ListIterator.hasPrevious())
        {
            Integer avg = ListIterator.previous();
            System.out.println(avg);
        }
    }
}
```

Output:

```
Iteration over the Stack from top to bottom:
988
203
119
```

IMPLEMENTATION

- We make a static array with size 100, which is the maximum size the Stack can have. Initially, our Stack is empty.
- The pseudo-code is presented in C-style.

```
struct CharStack1000
{
    char buffer[ 1000 ];
    int top = -1;
    // Default value of top is -1 when declaring the stack.
    // -1 means our stack is empty
};

void push( CharStack1000 &stack , char newElement )
{
    ++stack.top;
    stack.buffer[ stack.top ] = newElement;
}

char front( CharStack1000 &stack )
{
    return stack.buffer[ stack.top ];
}

void pop( CharStack1000 &stack )
{
    --stack.top;
}

int size( CharStack1000 &stack )
{
    return ( stack.top + 1 ); // simple
}

bool isEmptyStack( CharStack1000 &stack )
{
    return ( stack.top == -1 );
}
```

```java
import static java.lang.System.exit;
// Create Stack Using Linked list
class StackUsingLinkedlist {
    // A linked list node
    private class Node {
        int data; // integer data
        Node link; // reference variable Node type
    }
    // create global top reference variable global
    Node top;
    // Constructor
    StackUsingLinkedlist() {
        this.top = null;
    }
    // Utility function to add an element x in the stack
    public void push(int x) { // insert at the beginning
        // create new node temp and allocate memory
        Node temp = new Node();
        // check if stack (heap) is full. Then inserting an
        // element would lead to stack overflow
        if (temp == null) {
            System.out.print("\nHeap Overflow");
            return;
        }
        // initialize data into temp data field
        temp.data = x;
        // put top reference into temp link
        temp.link = top;
        // update top reference
        top = temp;
    }
    // Utility function to check if the stack is empty or not
    public boolean isEmpty() {
        return top == null;
    }
    // Utility function to return top element in a stack
    public int peek() {
        // check for empty stack
        if (!isEmpty()) {
            return top.data;
        } else {
            System.out.println("Stack is empty");
            return -1;
        }
    }
    // Utility function to pop top element from the stack
    public void pop() { // remove at the beginning
        // check for stack underflow
        if (top == null) {
            System.out.print("\nStack Underflow");
            return;
        }
        // update the top pointer to point to the next node
        top = (top).link;
    }
    public void display() {
        // check for stack underflow
        if (top == null) {
            System.out.printf("\nStack Underflow");
            exit(1);
        } else {
            Node temp = top;
            while (temp != null) {
                // print node data
                System.out.printf("%d->", temp.data);
                // assign temp link to temp
                temp = temp.link;
            }
        }
    }
}
public class GFG {
    public static void main(String[] args) {
        StackUsingLinkedlist obj = new StackUsingLinkedlist();
        obj.push(11);
        obj.push(22);
        obj.push(33);
        obj.push(44);
        // print Stack elements
        obj.display();
        // print Top element of Stack
        System.out.printf("\nTop element is %d\n", obj.peek());
        // Delete top element of Stack
        obj.pop();
        obj.pop();
        // print Stack elements
        obj.display();
        // print Top element of Stack
        System.out.printf("\nTop element is %d\n", obj.peek());
    }
}
```

Output:

```
44->33->22->11->
Top element is 44
22->11->
Top element is 22
```

OUTLINE

- Stack and Implementation
- Queue and Implementation
- Examples