Stack using Array - GeeksforGeeks

Source: https://www.geeksforgeeks.org/implement-stack-using-array/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Stack using Array Last Updated : 15 Dec, 2025 A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It can be implemented using an array by treating the end of the array as the top of the stack. A stack can be implemented using an array where we maintain: An integer array to store elements. A variable capacity to represent the maximum size of the stack. A variable top to track the index of the top element. Initially, top = -1 to indicate an empty stack. Try it on GfG Practice C++ class myStack { // array to store elements int * arr ; // maximum size of stack int capacity ; // index of top element int top ; public : // constructor myStack (int cap) { capacity = cap ; arr = new int [capacity]; top = -1 ; } }; C typedef struct { int * arr ; // array to store elements int capacity ; // maximum size of stack int top ; // index of top element } Stack ; // Function to create a stack (constructor equivalent) Stack * createStack (int capacity) { Stack * stack = (Stack *) malloc (sizeof (Stack)); stack -> capacity = capacity ; stack -> arr = (int *) malloc (capacity * sizeof (int)); stack -> top = -1 ; return stack ; } // Function to free stack memory void freeStack (Stack * stack) { free (stack -> arr); free (stack); } Java class myStack { // array to store elements private int [] arr ; // maximum size of stack private int capacity ; // index of top element private int top ; // constructor public myStack (int cap) { capacity = cap ; arr = new int [capacity] ; top = - 1 ; } } Python class myStack : def __init__ (self , cap): # array to store elements self . arr = [0] * cap # maximum size of stack self . capacity = cap # index of top element self . top = - 1 C# class myStack { // array to store elements private int [] arr ; // maximum size of stack private int capacity ; // index of top element private int top ; // constructor public myStack (int cap) { capacity = cap ; arr = new int [capacity]; top = - 1 ; } } JavaScript class myStack { // constructor constructor (cap) { // array to store elements this . arr = new Array (cap); // maximum size of stack this . capacity = cap ; // index of top element this . top = - 1 ; } } Operations On Stack Push Operation: Adds an item to the stack. If the stack is full, then it is said to be an Overflow condition. Before pushing the element to the stack, we check if the stack is full. If the stack is full (top == capacity-1) , then Stack Overflows and we cannot insert the element to the stack. Otherwise, we increment the value of top by 1 (top = top + 1) and the new value is inserted at top position . The elements can be pushed into the stack till we reach the capacity of the stack. C++ void push (int x) { if (top == capacity - 1) { cout << "Stack Overflow \n " ; return ; } arr [++ top] = x ; } C typedef struct { int * arr ; int capacity ; int top ; } Stack ; void push (Stack * stack , int x) { if (stack -> top == stack -> capacity - 1) { printf ("Stack Overflow \n "); return ; } stack -> arr [++ stack -> top] = x ; } Java void push (int x) { if (top == capacity - 1) { System . out . println ("Stack Overflow"); return ; } arr [++ top] = x ; } Python def push (self , x): if self . top == self . capacity - 1 : print ("Stack Overflow") return self . top += 1 self . arr [self . top] = x C# void push (int x) { if (top == capacity - 1) { Console . WriteLine ("Stack Overflow"); return ; } arr [++ top] = x ; } JavaScript function push (x) { if (top === capacity - 1) { console . log ('Stack Overflow'); return ; } arr [++ top] = x ; } Time Complexity: O(1) Auxiliary Space: O(1) Pop Operation: 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. Before popping the element from the stack, we check if the stack is empty . If the stack is empty (top == -1), then Stack Underflows and we cannot remove any element from the stack. Otherwise, we store the value at top, decrement the value of top by 1 (top = top – 1) and return the stored top value. C++ int pop () { if (top == -1) { cout << "Stack Underflow \n " ; return -1 ; } return arr [top --]; } C int pop () { if (top == -1) { cout << "Stack Underflow \n " ; return -1 ; } return arr [top --]; } Java int pop () { if (top == - 1) {

System . out . println ("Stack Underflow"); return - 1 ; } return arr [top --] ; } Python def pop (self): if self . top == - 1 : print ("Stack Underflow") return - 1 value = self . arr [self . top] self . top -= 1 return value C# int pop () { if (top == - 1) { Console . WriteLine ("Stack Underflow"); return - 1 ; } return arr [top --]; } JavaScript function pop () { if (top == - 1) { console . log ('Stack Underflow'); return - 1 ; } return arr [top --]; } Time Complexity: O(1) Auxiliary Space: O(1) Top or Peek Operation in Stack: Returns the top element of the stack. Before returning the top element from the stack, we check if the stack is empty. If the stack is empty (top == -1), we simply print “Stack is empty”. Otherwise, we return the element stored at index = top . C++ int peek () { if (top == -1) { cout << "Stack is Empty \n " ; return -1 ; } return arr [top]; } C int peek () { if (top == -1) { cout << "Stack is Empty \n " ; return -1 ; } return arr [top]; } Java int peek () { if (top == - 1) { System . out . println ("Stack is Empty"); return - 1 ; } return arr [top] ; } Python def peek (self): if self . top == - 1 : print ("Stack is Empty") return - 1 return self . arr [self . top] C# int peek () { if (top == - 1) { Console . WriteLine ("Stack is Empty"); return - 1 ; } return arr [top]; } JavaScript function peek () { if (top == - 1) { console . log ('Stack is Empty'); return - 1 ; } return arr [top]; } Time Complexity: O(1) Auxiliary Space: O(1) isEmpty Operation in Stack: Returns true if the stack is empty, else false. Check for the value of top in stack. If (top == -1) , then the stack is empty so return true. Otherwise, the stack is not empty so return false. C++ bool isEmpty () { return top == -1 ; } C bool isEmpty () { return top == -1 ; } Java boolean isEmpty () { return top == - 1 ; } Python def isEmpty (self): return self . top == - 1 C# bool isEmpty () { return top == - 1 ; } JavaScript isEmpty () { return this . top === - 1 ; } Time Complexity: O(1) Auxiliary Space: O(1) isFull Operation in Stack : Returns true if the stack is full, else false. Check for the value of top in stack. If (top == capacity-1), then the stack is full so return true. Otherwise, the stack is not full so return false. Time Complexity: O(1) Auxiliary Space: O(1) Full Implementation of Stack using Array C++ #include <iostream> using namespace std ; class myStack { // array to store elements int * arr ; // maximum size of stack int capacity ; // index of top element int top ; public : // constructor myStack (int cap) { capacity = cap ; arr = new int [capacity]; top = -1 ; } // push operation void push (int x) { if (top == capacity - 1) { cout << "Stack Overflow \n " ; return ; } arr [++ top] = x ; } // pop operation int pop () { if (top == -1) { cout << "Stack Underflow \n " ; return -1 ; } return arr [top --]; } // peek (or top) operation int peek () { if (top == -1) { cout << "Stack is Empty \n " ; return -1 ; } return arr [top]; } // check if stack is empty bool isEmpty () { return top == -1 ; } // check if stack is full bool isFull () { return top == capacity - 1 ; } }; int main () { myStack st (4); // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element cout << "Popped: " << st . pop () << " \n " ; // checking top element cout << "Top element: " << st . peek () << " \n " ; // checking if stack is empty cout << "Is stack empty: " << (st . isEmpty () ? "Yes" : "No") << " \n " ; // checking if stack is full cout << "Is stack full: " << (st . isFull () ? "Yes" : "No") << " \n " ; return 0 ; } C #include <stdbool.h> #include <stdio.h> #include <stdlib.h> // Define stack structure typedef struct { int * arr ; // array to store elements int capacity ; // maximum size of stack int top ; // index of top element } Stack ; // Function to create a stack Stack * createStack (int capacity) { Stack * stack = (Stack *) malloc (sizeof (Stack)); stack -> capacity = capacity ; stack -> arr = (int *) malloc (capacity * sizeof (int)); stack -> top = -1 ; return stack ; } // Push operation void push (Stack * stack , int x) { if (stack -> top == stack -> capacity - 1) { printf ("Stack Overflow \n "); return ; } stack -> arr [++ stack -> top] = x ; } // Pop operation int pop (Stack * stack) { if (stack -> top == -1) { printf ("Stack Underflow \n "); return -1 ; } return stack -> arr [stack -> top --]; } // Peek operation int peek (Stack * stack) { if (stack -> top == -1) { printf ("Stack is Empty \n "); return -1 ; } return stack -> arr [stack -> top]; } // Check if stack is empty bool isEmpty (Stack * stack) { return stack -> top == -1 ; } // Check if stack is full bool isFull (Stack * stack) { return stack -> top == stack -> capacity - 1 ; } // Free stack memory void freeStack (Stack * stack) { free (stack -> arr); free (stack); } int main () { Stack * st = createStack (4); // Push elements push (st , 1); push (st , 2); push (st , 3); push (st , 4); // Pop one element printf ("Popped: %d \n " , pop (st)); // Peek top element printf ("Top element: %d \n " , peek (st)); // Check if stack is empty printf ("Is stack empty: %s \n " , isEmpty (st) ? "Yes" : "No"); // Check if stack is full printf ("Is stack full: %s \n " , isFull (st) ? "Yes" : "No"); freeStack (st); // free memory return 0 ; } Java import java.util.Arrays ; class myStack { // array to store elements private int [] arr ; // maximum size of stack private int capacity ; // index of top element private int top ; // constructor public myStack (int cap) { capacity = cap ; arr = new int [capacity] ; top = - 1 ; } // push operation public void push (int x) { if (top == capacity - 1) { System . out . println ("Stack Overflow"); return ; } arr [++ top] = x ; } // pop operation public int pop () { if (top == - 1) { System . out . println ("Stack Underflow"); return - 1 ; } return arr [top --] ; } // peek (or top) operation public int peek () { if (top == - 1) { System . out . println ("Stack is Empty"); return - 1 ; } return arr [top] ; } // check if stack is empty public boolean isEmpty () { return top == - 1 ; } // check if stack is full public boolean isFull () {

return top == capacity - 1 ; } } public class Main { public static void main (String [] args) { myStack st = new myStack (4); // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element System . out . println ("Popped: " + st . pop ()); // checking top element System . out . println ("Top element: " + st . peek ()); // checking if stack is empty System . out . println ("Is stack empty: " + (st . isEmpty () ? "Yes" : "No")); // checking if stack is full System . out . println ("Is stack full: " + (st . isFull () ? "Yes" : "No")); } } Python class myStack : # constructor def __init__ (self , cap): self . capacity = cap self . arr = [0] * self . capacity self . top = - 1 # push operation def push (self , x): if self . top == self . capacity - 1 : print ("Stack Overflow") return self . top += 1 self . arr [self . top] = x # pop operation def pop (self): if self . top == - 1 : print ("Stack Underflow") return - 1 val = self . arr [self . top] self . top -= 1 return val # peek (or top) operation def peek (self): if self . top == - 1 : print ("Stack is Empty") return - 1 return self . arr [self . top] # check if stack is empty def isEmpty (self): return self . top == - 1 # check if stack is full def isFull (self): return self . top == self . capacity - 1 if __name__ == "__main__" : st = myStack (4) # pushing elements st . push (1) st . push (2) st . push (3) st . push (4) # popping one element print ("Popped:" , st . pop ()) # checking top element print ("Top element:" , st . peek ()) # checking if stack is empty print ("Is stack empty: " , "Yes" if st . isEmpty () else "No") # checking if stack is full print ("Is stack full: " , "Yes" if st . isFull () else "No") C# using System ; class myStack { // array to store elements private int [] arr ; // maximum size of stack private int capacity ; // index of top element private int top ; public myStack (int cap) { capacity = cap ; arr = new int [capacity]; top = - 1 ; } // push operation public void push (int x) { if (top == capacity - 1) { Console . WriteLine ("Stack Overflow"); return ; } arr [++ top] = x ; } // pop operation public int pop () { if (top == - 1) { Console . WriteLine ("Stack Underflow"); return - 1 ; } return arr [top --]; } // peek (or top) operation public int peek () { if (top == - 1) { Console . WriteLine ("Stack is Empty"); return - 1 ; } return arr [top]; } // check if stack is empty public bool isEmpty () { return top == - 1 ; } // check if stack is full public bool isFull () { return top == capacity - 1 ; } } class GfG { static void Main () { myStack st = new myStack (4); // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element Console . WriteLine ("Popped:" + st . pop ()); // checking top element Console . WriteLine ("Top element:" + st . peek ()); // checking if stack is empty Console . WriteLine ("Is stack empty:" + (st . isEmpty () ? "Yes" : "No")); // checking if stack is full Console . WriteLine ("Is stack full:" + (st . isFull () ? "Yes" : "No")); } } JavaScript class myStack { constructor (cap) { // array to store elements this . arr = new Array (cap); // maximum size of stack this . capacity = cap ; // index of top element this . top = - 1 ; } // push operation push (x) { if (this . top === this . capacity - 1) { console . log ("Stack Overflow"); return ; } this . arr [++ this . top] = x ; } // pop operation pop () { if (this . top === - 1) { console . log ("Stack Underflow"); return - 1 ; } return this . arr [this . top --]; } // peek (or top) operation peek () { if (this . top === - 1) { console . log ("Stack is Empty"); return - 1 ; } return this . arr [this . top]; } // check if stack is empty isEmpty () { return this . top === - 1 ; } // check if stack is full isFull () { return this . top === this . capacity - 1 ; } } // Driver Code let st = new myStack (4); // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element console . log ("Popped: " + st . pop ()); // checking top element console . log ("Top element: " + st . peek ()); // checking if stack is empty console . log ("Is stack empty: " + (st . isEmpty () ? "Yes" : "No")); // checking if stack is full console . log ("Is stack full: " + (st . isFull () ? "Yes" : "No")); Output Popped: 4 Top element: 3 Is stack empty: No Is stack full: No Stack Implementation using Dynamic Array When using a fixed-size array, the stack has a maximum capacity that cannot grow beyond its initial size. To overcome this limitation, we can use dynamic arrays . Dynamic arrays automatically resize themselves as elements are added or removed, which makes the stack more flexible. In C++, we can use vector. In Java, we can use ArrayList. In Python, we can use list. In C#, we can use List. In JavaScript, we can use Array. C++ #include <iostream> #include <vector> using namespace std ; class myStack { vector < int > arr ; public : // push operation void push (int x) { arr . push_back (x); } // pop operation int pop () { if (arr . empty ()) { cout << "Stack Underflow" << endl ; return -1 ; } int val = arr . back (); arr . pop_back (); return val ; } // peek operation int peek () { if (arr . empty ()) { cout << "Stack is Empty" << endl ; return -1 ; } return arr . back (); } // check if stack is empty bool isEmpty () { return arr . empty (); } // current size int size () { return arr . size (); } }; int main () { myStack st ; // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element cout << "Popped: " << st . pop () << endl ; // checking top element cout << "Top element: " << st . peek () << endl ; // checking if stack is empty cout << "Is stack empty: " << (st . isEmpty () ? "Yes" : "No") << endl ; // checking current size cout << "Current size: " << st . size () << endl ; } C #include <stdbool.h> #include <stdio.h> #include <stdlib.h> typedef struct { int * arr ; // dynamic array to store stack elements int top ; // index of top element int capacity ; // current capacity of the stack } Stack ; // Initialize stack Stack * createStack (int initialCapacity) { Stack * stack = (Stack *) malloc (sizeof (Stack)); stack -> arr = (int *) malloc (initialCapacity * sizeof (int)); stack -> top = -1 ; stack -> capacity = initialCapacity ; return stack ; } // Push operation void push (Stack * stack , int value) { // Resize array if needed if (stack -> top + 1 >= stack -> capacity) { stack -> capacity *= 2 ; // double the capacity stack -> arr = (int *) realloc (stack -> arr , stack -> capacity * sizeof (int)); } stack -> arr [++ stack -> top] = value ; } // Pop operation int pop (Stack * stack) { if (stack -> top == -1) { printf ("Stack Underflow \n "); return -1 ; } return stack -> arr [stack -> top --]; } // Peek operation int peek (Stack * stack) { if (stack -> top == -1) { printf ("Stack is Empty \n "); return -1 ; } return stack -> arr [stack -> top]; } // Check if stack is empty bool isEmpty (Stack * stack) { return stack -> top == -1 ; } // Current size of stack int size (Stack * stack) { return stack -> top + 1 ; } // Free stack memory void freeStack (Stack * stack) { free (stack -> arr); free (stack); } int main () { Stack * st = createStack (2); // initial capacity 2 // Push elements push (st , 1); push (st , 2); push (st , 3); push (st , 4); // Pop one element printf ("Popped: %d \n " , pop (st)); // Peek top element printf ("Top element: %d \n " , peek (st)); // Check if stack is empty printf ("Is stack empty: %s \n " , isEmpty (st) ? "Yes" : "No"); // Current size printf ("Current size: %d \n " , size (st)); freeStack (st); // free memory return 0 ; } Java import java.util.ArrayList ; class myStack { ArrayList < Integer > arr = new ArrayList <> (); // push operation void push (int x) { arr . add (x); } // pop operation int pop () { if (arr . isEmpty ()) { System . out . println ("Stack Underflow"); return - 1 ; } int val = arr . get (arr . size () - 1); arr . remove (arr . size () - 1); return val ; } // peek operation int peek () { if (arr . isEmpty ()) { System . out . println ("Stack is Empty"); return - 1 ; } return arr . get (arr . size () - 1); } // check if stack is empty boolean isEmpty () { return arr . isEmpty (); } // current size int size () { return arr . size (); } public static void main (String [] args) { myStack st = new myStack (); // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element System . out . println ("Popped: " + st . pop ()); // checking top element System . out . println ("Top element: " + st . peek ()); // checking if stack is empty System . out . println ("Is stack empty: " + (st . isEmpty () ? "Yes" : "No")); // checking current size System . out . println ("Current size: " + st . size ()); } } Python class myStack : def __init__ (self): self . arr = [] # push operation def push (self , x): self . arr . append (x) # pop operation def pop (self): if not self . arr : print ("Stack Underflow") return - 1 return self . arr . pop () # peek operation def peek (self): if not self . arr : print ("Stack is Empty") return - 1 return self . arr [- 1] # check if stack is empty def isEmpty (self): return len (self . arr) == 0 # current size def size (self): return len (self . arr) if __name__ == "__main__" : st = myStack () # pushing elements st . push (1) st . push (2) st . push (3) st . push (4) # popping one element print ("Popped:" , st . pop ()) # checking top element print ("Top element:" , st . peek ()) # checking if stack is empty print ("Is stack empty:" , "Yes" if st . isEmpty () else "No") # checking current size print ("Current size:" , st . size ()) C# using System ; using System.Collections.Generic ; class myStack { List < int > arr = new List < int > (); // push operation public void push (int x) { arr . Add (x); } // pop operation public int pop () { if (arr . Count == 0) { Console . WriteLine ("Stack Underflow"); return - 1 ; } int val = arr [arr . Count - 1]; arr . RemoveAt (arr . Count - 1); return val ; } // peek operation public int peek () { if (arr . Count == 0) { Console . WriteLine ("Stack is Empty"); return - 1 ; } return arr [arr . Count - 1]; } // check if stack is empty public bool isEmpty () { return arr . Count == 0 ; } // current size public int size () { return arr . Count ; } public static void Main () { myStack st = new myStack (); // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element Console . WriteLine ("Popped: " + st . pop ()); // checking top element Console . WriteLine ("Top element: " + st . peek ()); // checking if stack is empty Console . WriteLine ("Is stack empty: " + (st . isEmpty () ? "Yes" : "No")); // checking current size Console . WriteLine ("Current size: " + st . size ()); } } JavaScript class myStack { constructor () { this . arr = []; } // push operation push (x) { this . arr . push (x); } // pop operation pop () { if (this . arr . length === 0) { console . log ("Stack Underflow"); return - 1 ; } return this . arr . pop (); } // peek operation peek () { if (this . arr . length === 0) { console . log ("Stack is Empty"); return - 1 ; } return this . arr [this . arr . length - 1]; } // check if stack is empty isEmpty () { return this . arr . length === 0 ; } // current size size () { return this . arr . length ; } } // Driver Code let st = new myStack (); // pushing elements st . push (1); st . push (2); st . push (3); st . push (4); // popping one element console . log ("Popped:" , st . pop ()); // checking top element console . log ("Top element:" , st . peek ()); // checking if stack is empty console . log ("Is stack empty:" , st . isEmpty () ? "Yes" : "No"); // checking current size console . log ("Current size:" , st . size ()); Output Popped: 4 Top element: 3 Is stack empty: No Current size: 3 Fixed Size Stack Vs Dynamic Size Stack Comment Article Tags: Article Tags: Stack DSA Data Structures-Stack