# Lab Report-08 :

# Implementation of Stack Using Linked List

## Introduction

Implementation a stack data structure using a linked list in this lab. A stack is a fundamental data structure that follows the Last-In-First-Out (LIFO) principle, making it suitable for various applications such as expression evaluation, function call management, and more. The primary objective of this lab was to design and implement a stack data structure using a Array and linked list.

## Objectives:

The main objectives of this lab were as follows:

1. To understand the concepts of stacks and linked lists.
2. To design and implement a stack using a linked list and Array.
3. To perform basic stack operations such as push pop.
4. To analyze and compare stack operations' time and space complexity.

**Theory:**

A stack is a linear data structure that operates on the principle of Last-In-First-Out (LIFO). In a stack, elements are added and removed from the same end, known as the "top" of the stack. The primary operations associated with a stack are:

1. **Push**: Adding an element to the top of the stack.
2. **Pop**: Removing the top element from the stack.
3. **IsEmpty**: Checking if the stack is empty.

A linked list is a data structure consisting of a sequence of elements, each containing a reference (or link) to the next element in the sequence. Linked lists can be singly linked (each element points to the next one) or doubly linked (each element points to both the next and previous elements).

Key characteristics of a linked list include:

* **Node**: Each element in the linked list is called a node. A node typically contains two fields: a data element and a reference (link) to the next node.
* **Head**: The first node in the list is called the head.
* **Tail**: The last node in the list is called the tail.

**TASK-01:**

**Stack Implementation Using Array.**

In contrast to a linked list, a stack can also be implemented using an array. In this implementation:

* An array is used to store the elements of the stack.
* An index or pointer is used to keep track of the top element of the stack.
* Push operation increments the pointer and inserts an element at the corresponding index.
* Pop operation removes the element at the top index and decrements the pointer.
* Peek operation accesses the data at the top index without removing it.
* Is Empty operation checks if the stack is empty by examining the pointer value.

Using an array to implement a stack can have advantages regarding memory management and simplicity, but it may have limitations in terms of dynamic resizing.

**TASK-02:**

**Stack Implementation Using Linked List.**

Implementing a stack using a linked list involves utilizing the structure and operations of a linked list to mimic the behavior of a stack. In this implementation:

* The head of the linked list is the top of the stack.
* Push operation inserts a new node at the head of the list.
* Pop operation removes the node at the head of the list.
* IsEmpty operation checks if the linked list (stack) is empty by examining if the head is null.

This combination leverages the dynamic memory allocation and flexibility of linked lists while maintaining the LIFO behavior of a stack.

**TASK-01:** **Stack Implementation Using Array.**

**CODE:**

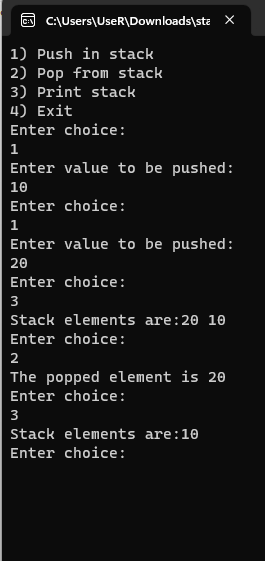
|  |  |
| --- | --- |
| #include <iostream>using namespace std;  int stack[100], n=100, top=-1;  void push(int val) {  if(top>=n-1)  cout<<"Stack Overflow"<<endl;  else {  top++;  stack[top]=val;  }  }  void pop() {  if(top<=-1)  cout<<"Stack Underflow"<<endl;  else {  cout<<"The popped element is "<< stack[top] <<endl;  top--;  }  }  void print\_stack() {  if(top>=0) {  cout<<"Stack elements are:";  push(val);  break;  }  case 2: {  pop();  break;  }  case 3: {  print\_stack();  break;  }  case 4: {  cout<<"Exit"<<endl;  break;  }  default: {  cout<<"Invalid Choice"<<endl;  }  }  }while(ch!=4);  return 0;  } | for(int i=top; i>=0; i--)  cout<<stack[i]<<" ";  cout<<endl;  } else  cout<<"Stack is empty";  }  int main() {  int ch, val;  cout<<"1) Push in stack"<<endl;  cout<<"2) Pop from stack"<<endl;  cout<<"3) Print stack"<<endl;  cout<<"4) Exit"<<endl;  do {  cout<<"Enter choice: "<<endl;  cin>>ch;  switch(ch) {  case 1: {  cout<<"Enter value to be pushed:"<<endl;  cin>>val; |

**TASK-02**: **Stack Implementation Using Linked List.**

**CODE:**

|  |  |
| --- | --- |
| #include <bits/stdc++.h>  using namespace std;  class Node  {  public:  int val;  Node \*next;  Node \*prev;  Node(int val)  {  this->val = val;  this->next = NULL;  this->prev = NULL;  }  };  class myStack  {  public:  Node \*head = NULL;  Node \*tail = NULL;  int sz = 0;  void push(int val)  {  sz++;  Node \*newNode = new Node(val);  if (head == NULL)  {  head = newNode;  tail = newNode;  return;  }  newNode->prev = tail;  tail->next = newNode;  tail = tail->next;  }  void pop()  {  sz--;  Node \*deleteNode = tail;  tail = tail->prev;  if (tail == NULL)  {  head = NULL; | }  else  {  tail->next = NULL;  }  delete deleteNode;  }  int top()  return tail->val;  int size()  {  return sz;  }  bool empty()  {  if (sz == 0)  return true;  else  return false;  }  };  int main()  {  myStack st;  // st.pop();  // cout << st.top() << endl;  int n;  cin >> n;  for (int i = 0; i < n; i++)  {  int x;  cin >> x;  st.push(x);  }  while (!st.empty())  {  cout << st.top() << endl;  st.pop();  }  return 0;  } |

**OUTPUT- task-1**



**OUTPUT- task-2**

