Faculty of Computing

Logo, company name

Description automatically generated

Data Structure and Algorithm

Lab Manual

**Lab 7: Stack Data Structure**

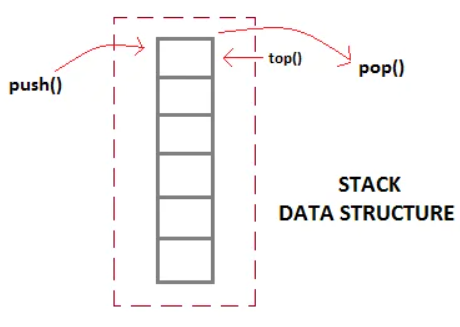
**Objectives:**

* Stack
* Stack using Linked List Concept
* Stack using Array Concept

**Introduction**

Suppose that you have a program with several functions. To be specific, suppose that you have the functions A, B, C, and D in your program. Now suppose that function A calls function B, function B calls function C, and function C calls function D. When function D terminates, control goes back to function C; when function C terminates, control goes back to function B; and when function B terminates, control goes back to function A. During program execution, how do you think the computer keeps track of the function calls? What about recursive functions? How does the computer keep track of the recursive calls?

This section discusses the data structure called the stack, which the computer uses to implement function calls. You can also use stacks to convert recursive algorithms into non recursive algorithms, especially recursive algorithms that are not tail recursive. Stacks have numerous other applications in computer science. After developing the tools necessary to implement a stack, we will examine some applications of stacks.



Stack data structure(LIFO)

**Activity Time boxing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task No.** | **Activity Name** | **Activity time** | **Total Time** |
| 1 | Lab Manual Lecture | 30 mins |  |
| 2. | Example | 10 mins |  |
| 3 | Walkthrough Tasks | 10 mins |  |
| 4. | Lab Tasks | 90 mins |  |
| 5. | Evaluation | 30 mins | 170 mins |

**Concept Map**

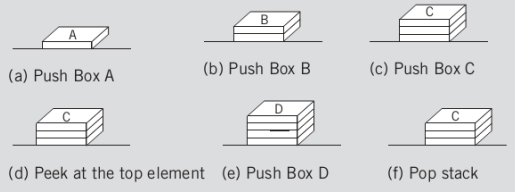
**Stack (LIFO)**

A stack is a list of homogenous elements in which the addition and deletion of elements occurs only at one end, called the top of the stack. The elements at the bottom of the stack have been in the stack the longest. The top element of the stack is the last element added to the stack. Because the elements are added and removed from one end (that is, the top), it follows that the item that is added last will be removed first. For this reason, a stack is also called a Last In First Out (LIFO) data structure.

**Push () and Pop() working:** The push, top, and pop operations work as follows: Suppose there are boxes lying on the floor that need to be stacked on a table. Initially, all of the boxes are on the floor and the stack is empty.



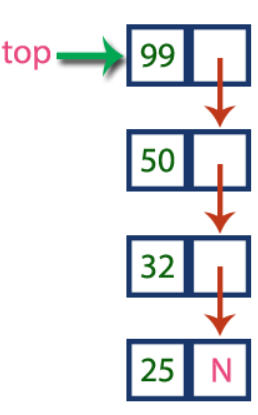
Different push and pop will make this stack looks as:



**Stack Using Link List:**

Stacks can be easily implemented using a linked list. Stack is a data structure to which a data can be added using the push() method and data can be removed from it using the pop() method. With Linked list, the push operation can be replaced by the addAtFront() method of linked list and pop operation can be replaced by a function which deletes the front node of the linked list. In this way our Linked list will virtually become a Stack with push() and pop() methods.

In the below example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32,50 and 99.



**Sample Program:**

// C++ program to Implement a stack

// using singly linked list

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class Stack {

private:

Node\* top;

public:

Stack() {

top = nullptr;

}

void push(int value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->next = top;

top = newNode;

cout << value << " pushed into stack." << endl;

}

void pop() {

if (isEmpty()) {

cout << "Stack Underflow! Cannot pop." << endl;

return;

}

Node\* temp = top;

cout << top->data << " popped from stack." << endl;

top = top->next;

delete temp;

}

bool isEmpty() {

return top == nullptr;

}

void display() {

if (isEmpty()) {

cout << "Stack is empty!" << endl;

return;

}

cout << "Stack elements: ";

Node\* temp = top;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

~Stack() {

//while (!isEmpty()) {

//pop();

}

};

int main() {

Stack s;

s.push(5);

s.push(10);

s.push(15);

s.display();

// ✅ Pop only the first (top) element once

s.pop();

s.display();

return 0;

}

**Stack Using Arrays:**

A stack data structure can be implemented using a one-dimensional array. But stack implemented using array stores only a fixed number of data values. This implementation is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable called 'top'. Initially, the top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

**Sample Program:**

/\* C++ program to implement basic stack

operations \*/

#include <bits/stdc++.h>

using namespace std;

#define MAX 1000

class Stack {

int top;

public:

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

Stack() { top = -1; }

bool push(int x);

int pop();

int peek();

bool isEmpty();

};

bool Stack::push(int x)

{

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

cout << "Stack Overflow";

return false;

}

else {

a[++top] = x;

cout << x << " pushed into stack\n";

return true;

}

}

int Stack::pop()

{

if (top < 0) {

cout << "Stack Underflow";

return 0;

}

else {

int x = a[top--];

return x;

}

}

int Stack::peek()

{

if (top < 0) {

cout << "Stack is Empty";

return 0;

}

else {

int x = a[top];

return x;

}

}

bool Stack::isEmpty()

{

return (top < 0);

}

// Driver program to test above functions

int main()

{

class Stack s;

s.push(10);

s.push(20);

s.push(30);

cout << s.pop() << " Popped from stack\n";

//print top element of stack after poping

cout << "Top element is : " << s.peek() << endl;

//print all elements in stack :

cout <<"Elements present in stack : ";

while(!s.isEmpty())

{

// print top element in stack

cout << s.peek() <<" ";

// remove top element in stack

s.pop();

}

return 0;

}**/\* 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;

}

}

void print(){

for(int i = top;i>-1;i--){

System.out.print(" "+ a[i]);

}

}

}

// 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");

System.out.println("Top element is :" + s.peek());

System.out.print("Elements present in stack :");

s.print();

}

}

**Practice Tasks**

**[Estimated Time]**

**6.1 Task 1: [Time Required: 30 minutes]**

Create a classNode for book that contains information about title, price, edition, and no of pages of the

book. Implement your push, pop, peek books on a stack functions properly.

Now in the main:

1. Push 5 books into this stack.

2. Find the top element of the stack

3. Pop 2 books from the stack.

4. Display all the remaining books in the stack.

### Lab Task 2 [Time required: 1 hour]

Design an inventory class that stores the following members:

serialNum: An integer that holds a part's serial number.

manufactYear: An integer that holds the year the part was manufactured.

lotNum: An integer that holds the part's lot number.

The class should have appropriate member functions for storing data into, and retrieving data from, these members.

Next, design a stack class that can hold objects of the class described above.

Last, design a program that uses the stack class described above.

The program should have a loop that asks the user if he or she wishes to add a part to inventory, or take a part from inventory. The loop should repeat until the user is finished.

If the user wishes to add a part to inventory, the program should ask for the serial number, year of manufacture, and lot number.

The data should be stored in an inventory object, and pushed onto the stack.

If the user wishes to take a part from inventory, the program should pop the top-most part from the stack and display the contents of its member variables.

When the user finishes the program, it should display the contents of the member values of all the objects that remain on the stack. [implement using linklist]

**Evaluation criteria**

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

**Further Reading**

The slides and reading material can be accessed from the folder of the class instructor available at Moellim.

**Out comes**

The outcomes of this lab were:

* Stack
* Stack using Linked List Concept
* Stack using Array Concept