***Introduction:***

**Stack** is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the **top** of the stack and the only element that can be removed is the element that is at the top of the stack, just like a pile of objects.

* Stack is an **ordered list** of **similar data type**.
* Stack is a **LIFO**(Last in First out) structure or we can say **FILO**(First in Last out).
* push() function is used to insert new elements into the Stack and pop() function is used to remove an element from the stack. Both insertion and removal are allowed at only one end of Stack called **Top**.
* Stack is said to be in **Overflow** state when it is completely full and is said to be in **Underflow** state if it is completely empty.

***Theory:***

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.

Stack can be implemented in two ways in C:

1. Using Arrays.
2. Using Linked List.

***Objective:***

At the end of this experiment you will be able to perform :

* Stack using array in C.
* Stack using Linked List in C.

***Experiment:***

Stack Using Arrays in C:

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

struct StackNode {

int data;

struct StackNode\* next;

};

struct StackNode\* newNode(int data)

{

struct StackNode\* stackNode = (struct StackNode\*)malloc(sizeof(struct StackNode));

stackNode->data = data;

stackNode->next = NULL;

return stackNode;

}

int isEmpty(struct StackNode\* root)

{

return !root;

}

void push(struct StackNode\*\* root, int data)

{

struct StackNode\* stackNode = newNode(data);

stackNode->next = \*root;

\*root = stackNode;

printf("%d pushed to stack\n", data);

}

int pop(struct StackNode\*\* root)

{

if (isEmpty(\*root))

return INT\_MIN;

struct StackNode\* temp = \*root;

\*root = (\*root)->next;

int popped = temp->data;

free(temp);

return popped;

}

int peek(struct StackNode\* root)

{

if (isEmpty(root))

return INT\_MIN;

return root->data;

}

int main()

{

struct StackNode\* root = NULL;

push(&root, 10);

push(&root, 20);

push(&root, 30);

printf("%d popped from stack\n", pop(&root));

printf("Top element is %d\n", peek(root));

return 0;

}

Stack Using Linked List in C:

void push ()

{

int val;

struct node \*ptr =(struct node\*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("not able to push the element");

}

else

{

printf("Enter the value");

scanf("%d",&val);

if(head==NULL)

{

ptr->val = val;

ptr -> next = NULL;

head=ptr;

}

else

{

ptr->val = val;

ptr->next = head;

head=ptr;

}

printf("Item pushed");

}

}

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n");

}

else

{

printf("Printing Stack elements \n");

while(ptr!=NULL)

{

printf("%d\n",ptr->val);

ptr = ptr->next;

}

}

}

***Algorithm:***

### Algorithm for PUSH operation:

1. Check if the stack is **full** or not.
2. If the stack is full, then print error of overflow and exit the program.
3. If the stack is not full, then increment the top and add the element.

### Algorithm for POP operation:

1. Check if the stack is empty or not.
2. If the stack is empty, then print error of underflow and exit the program.
3. If the stack is not empty, then print the element at the top and decrement the top.