**Stack**

A stack is a container of objects that are inserted and removed according to the last in first-out (LIFO) principle. It is used with set of algorithm lie the element inserted last deleted first we can get that number at first.

Ex: A stack of books; we can remove only the top book, also we can add a new book on the top.



**Operation performed on stack:**

1. Creation of stack
2. Push elements (Insertion of element into the stack)
3. Pop elements (Delete or collected the top most element)
4. Traversing element (To visit each element in the stack one by one)

**Function used in stack implementation:**

1. Push (int element);
2. Pop ();
3. Traverse ();
4. Peek();
5. isEmpty();
6. isFull();

A variable **top** is used to indicate the top element or the position where the element is to be inserted.

In Push operation we have to pass the element to be pushed into the stack. But in Pop operation there is no need to pass any element or address because it will always delete the element present at the top of the element.

**Ex:** 

We can implement the stack using the concept of array as well as linked list. We will first see about implementation of stack using array.

**Implementation of stack using array:**

1. Creation of array:

As we are using array concept, so we have to create an array and size is given to it. For size of array we have to use macro (preprocessor directive) because using variable within the square bracket is not allowed in c. Here a variable top is used whose initial value is -1 i.e. when the stack is empty. When the stack having some element top will hold the index of top most element. By this variable we can do many more operation.

#define CAPACITY 5

Int top = -1;

Int stack[capacity];

1. Insert elements into the array:

For inserting elements into stack we have to go for Push() method. Before pushing any element into the stack first we have to check whether the stack is capable of holding a new element. We have to check the stack is full or not.

void Push(int element) {

if(isFull()){

printf(“Stack Overflow!”);

}else{

top++;

stack[top]=element;

} }

Int isFull(){

If (top == CAPACITY-1) {

return 1;

} else {

return 0;

} }

1. Pop elements from stack:

For collecting elements from the stack we have to go for Pop() method. Before collecting any element from the stack first we have to check whether the stack is having any element or not. We have to check the stack is empty or not.

Int pop(){

int element;

if(isEmpty){

return 0;

} else {

element = stack[top];

top--;

}

return top; }

int isEmpty(){

if (top == -1){

return 1;

}else{

Return 0; }

}

1. Peek() method:

It is a method used to get or access the top element not remove. The difference between **Peek** and **Pop** is Pop will remove the top element but Peek will only return the top element not remove**.** Here before peeking the data we have to check the stack is empty or having some element.

Int Peek () {

If(isFull()){

return 0; }

else{

return stack[top];

}

1. Traverse() :

Here we are visiting each element and printing all the element present. Here also we have check the stack is having some element or it is empty.

void traverse () {

if(isEmpty())

printf(“Stack is Empty !”);

}

else{

Printf(“Stack elements are:”);

For(i=0;i<top;i++){

Printf(“%d”,stack[i]);

} } }

Ex: Program Implementing through stack

// Wap to implement stack using array

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define MAX 5

int stack\_arr[MAX];

int top=-1;

void push(int item);

int pop();

int peek();

int isEmpty();

int isFull();

void display();

int main(){

int item,choice;

while(1){

printf("1 . Push\n");

printf("2 . Pop\n");

printf("3 . Display top element\n");

printf("4 . Display stack elements\n");

printf("5 . Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice){

case 1:

printf("Enter the item to be pushed : ");

scanf("%d",&item);

push(item);

break;

case 2:

item = pop();

printf("Popped item is : %d\n",item);

break;

case 3:

printf("Item at the top of stack : %d\n",peek());

break;

case 4:

display();

break;

case 5:

exit(1);

default:

printf("Invalid Choice!");

break;

}

}

return 0;

}

void push(int item){

if(isFull()){

printf("Opps! Stack Overflow Occurs...");

return;

}

top += 1;

stack\_arr[top] = item;

}

int isFull(){

if(top == MAX-1){

return 1;

}

return 0;

}

int pop(){

if(isEmpty()){

printf("Opps! Stack underflow....");

exit(1);

}

int item;

item = stack\_arr[top];

top -= 1;

return item;

}

int isEmpty(){

if(top == -1){

return 1;

}

return 0;

}

int peek(){

if(isEmpty()){

printf("Stack underflow...");

exit(1);

}

return stack\_arr[top];

}

void display(){

int i;

if(isEmpty()){

printf("Stack is empty!");

return;

}

printf("Stack elements :");

for(i=top;i>=0;i--){

printf("%d\n",stack\_arr[i]);

printf("\n");

}

}

**Stack using single linked list:**

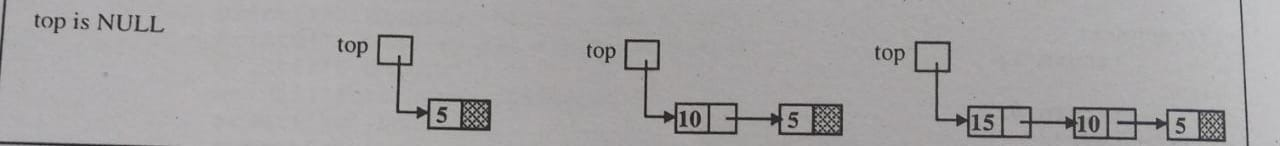
Here we are using the linked list to implement stack. Here we have to follow last in first out algorithm. So all the operation should be performed using top variable which is the last element inserted.

**Operation performed on stack:**

* **Insert or push() :**

When the stack is empty the top value is **null** because it is not pointing to any node. That is the case of underflow. When a node is inserted then top should point to the node. Inside the node some value should be present in data part and the link part should contain null because it is the last node in the stack.

When another node is inserted into the stack the **top** should point to the new node and new node should point to the previous node.



Code:

//Creation of node

struct node {

int data ; // Data part of the node

struct node\* link ; // Link part of the node

};

struct node\* top = null;

// top is a struct node pointer variable initializes with the value null

// the above part common for all operation logic for push operation starts from below

void push(){

struct node\* temp;

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

printf(“Enter node data: ”);

scanf(“%d”,&temp->data);

temp->link = top;

top = temp;

}

Explanation:

Inside the push function we created a pointer variable temp. The temp should point to the newly created node. Then we have to enter the data and store it into data part of node by temp->data. Now the value of top is stored into link part of node by temp->link = top; As top should always point to the newly created array so this can be done by top = temp;

* **Delete or pop () operation :**

Here we have to pop() the stack elements.

Algorithm:

First we have to check the stack is empty or having some element. If some element are present then only we can perform pop() operation. Then we de-reference the last node present at the top position and top should point to the next node.

Code:

void pop () {

struct node\* temp;

if(top == null){

printf(“Stack UNDERFLOW”);

exit(0);

}else{

temp = top;

top = top->link;

temp->link == null;

printf(“% number poped!”, temp->data);

}

Free(temp);

}

Explanation :

We have to check first whether the stack is having some element or not. If it is not empty then we should go for pop operation. We created a pointer variable which should point to the first node(node to be deleted ). Here we are de-referencing the top element by top = top -> link Then we have to store the link part as null by temp -> link = null After this we have to release the node from memory by Free(temp)

* **Traverse a stack:**

Traversing means visiting each node one by one. We can do many more things by traversing each node i.e. display each element, we can perform linear search etc.

void display () {

struct node\* temp;

if(top == null) {

printf(“Stack UNDERFLOW”);

exit(0);

}

temp = top;

while(temp != null){

print(“%d \n”, temp -> data);

temp = temp -> link;

}

}