Lab Work 9

Learning Objectives

- Introduction to stack
- Basic stack operations
- Introduction to queues
- Basic Queues operations

Introduction to stack

1. Stack is a data structure which follows LIFO i.e. Last-In-First-Out method.
2. The data/element which is stored last in the stack i.e. the element at top will be accessed first.
3. Both insertion & deletion takes place at the top.
4. When we implement stack uisng array we take the direction of the stack i.e the direction in which elements are inserted towards right.

Operations:

- isempty() - to check if the Stack is empty or not.
- push() - to insert element in the Stack.
- pop() - to delete element from stack.
- show_top() - to display element at top.

Stack implementation using array

Lets take an example of an array of 5 elements to implement stack. So we define the size of the array using pre-processor directive #deifne SIZE 5 & then we can create an array of 5 elements as int A[SIZE]; Also we will declare a top variable to track the element at the top of the stack which will initially be -1 when the stack is empty i.e int top=-1;

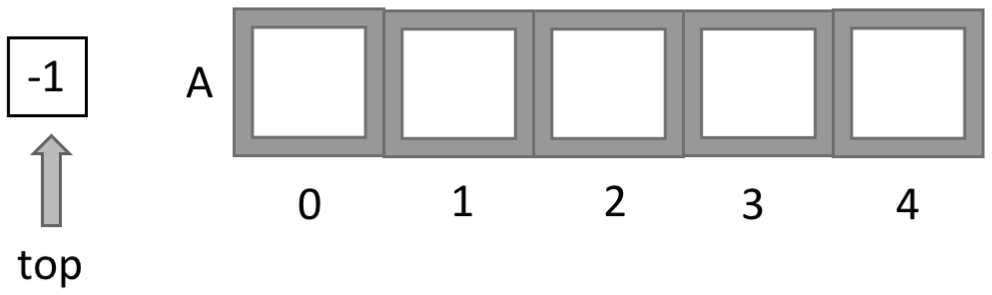


Figure 1: When the stack is empty. The top variable will point to -1 indicating that the stack is empty

isempty()- Now that we know that the stack is empty when top is equal to -1 we can use this to implement our isempty() function i.e. when top == -1 retrun true else return false.

push()- To push an element into the stack we will simply increment top by one and insert the element at that position. While inserting we have to take care of the condition when the array is full i.e. when top == SIZE-1;

pop()- To pop an element from the stack we will simple decrement top by one which will simply mean that the element is no longer the part of the stack. In this case we have to take care of the condition when the stack is empty i.e top == -1 then we cannot perform the pop operation.

show_top()- we will simply print the element at top if the stack is not empty.

Lets say we have an stack with one element i.e 2 and we have to insert ot push an element 3 to this stack.

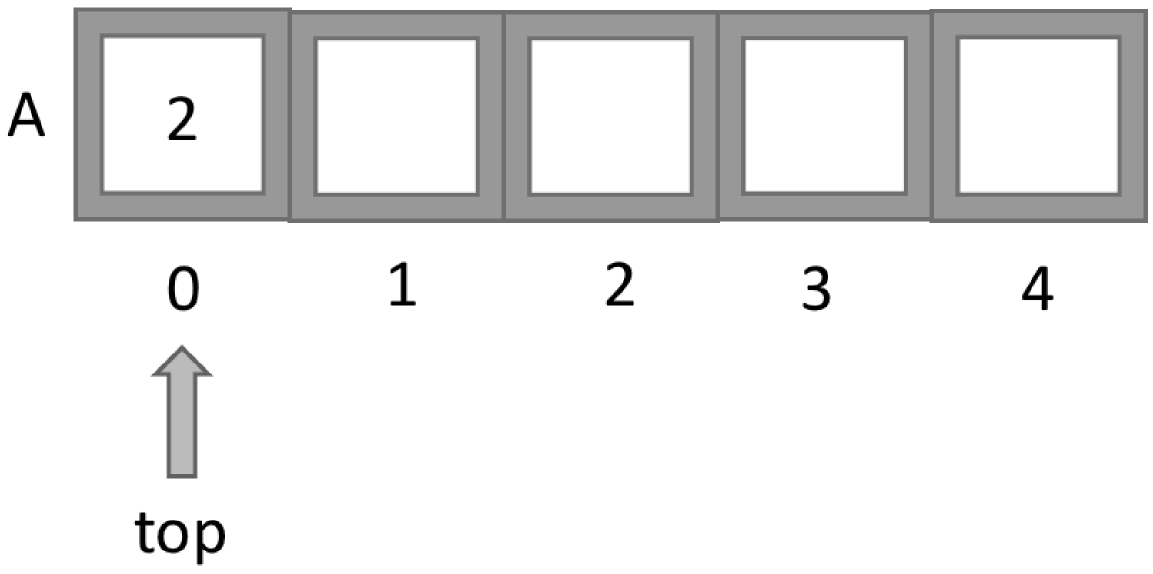


Figure 2: adding the first element in the stack and updating the top variable value

- then we will simply increment top by one and push the element at top.

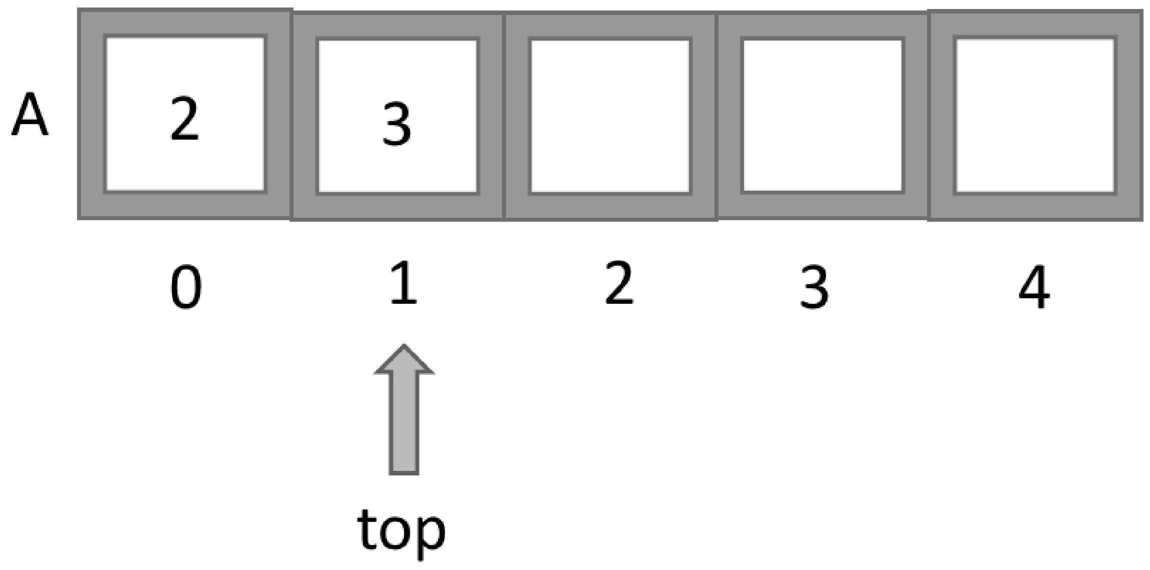


Figure 3: Adding the second element in the stack and updating the top variable value

- And to pop an element we will simply decrement top by one.

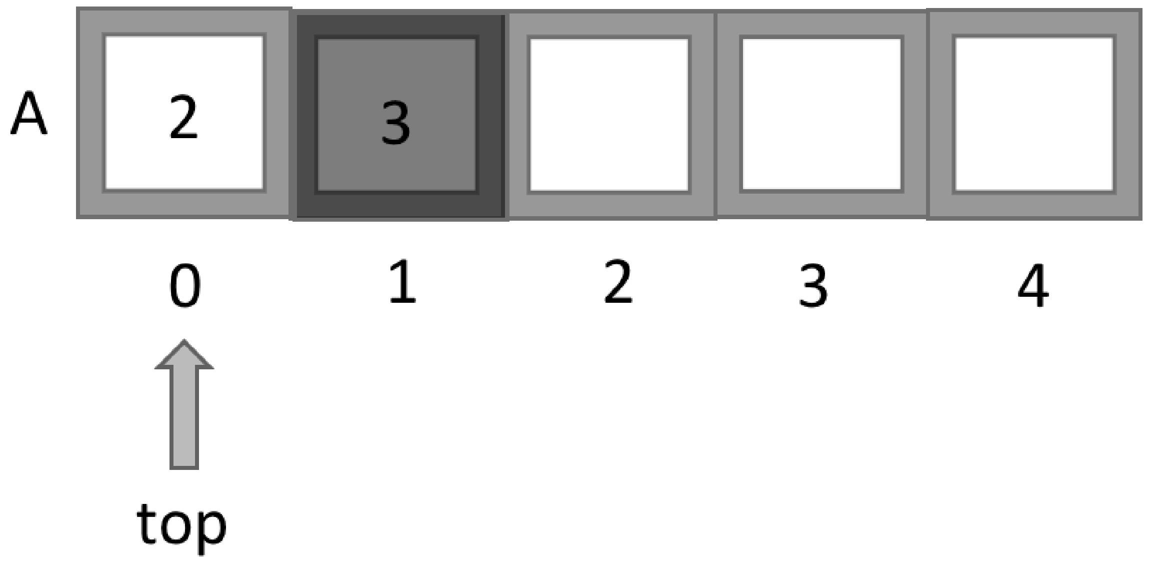


Figure 4: Popping the element

Example Code

```
#include <iostream>
using namespace std;

#define SIZE 5
int A[SIZE];
int top = -1;

bool isempty()
{
  if(top==-1)
  return true;
  else
  return false;
}

void push(int value)
{
  if(top==SIZE-1)
  {    cout<<"Stack is full!\n";
  }
   else
  {
    top++;
    A[top]=value;
  }
}

void pop()
{
 if(isempty())
  cout<<"Stack is empty!\n";
 else
  top--;
}

void show_top()
{
 if(isempty())
  cout<<"Stack is empty!\n";
 else
  cout<<"Element at top is: "<<A[top]<<"\n";

}

void displayStack()
{
  if(isempty())
 {
    cout<<"Stack is empty!\n";
 }
 else
 {
  for(int i=0 ; i<=top; i++)
    cout<<A[i]<<" ";
   cout<<"\n";

  }
}
int main()
{
 int choice, flag=1, value;
 while( flag == 1)
 {
 cout<<"\n1.PUSH 2.POP 3.SHOW_TOP 4.DISPLAY_STACK 5.EXIT\n";
 cin>>choice; switch (choice)
 {
 case 1: cout<<"Enter Value:\n";
         cin>>value;
         push(value);
         break;
 case 2: pop();
         break;
 case 3: show_top();
         break;
 case 4: displayStack();
         break;
 case 5: flag = 0;
         break;
 }
 }
  return 0;
}
```

Stack implementation using linked list

Stack is a data structure which follows LIFO i.e. Last-In-First-Out method.

The data/element which is stored last in the stack i.e. the element at top will be accessed first.

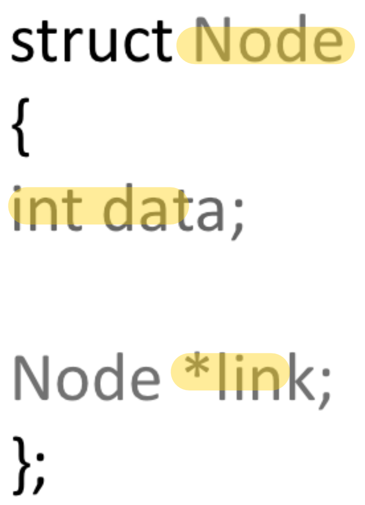
And both insertion & deletion takes place at the top.

When we implement stack using array :

1. We create an array of predefined size & we cannot increase the size of the array if we have more elements to insert.
2. If we create a very large array, we will be wasting a lot of memory space.

3. So to solve this lets try to implement Stack using Linked list where we will dynamically increase the size of the stack as per the requirement.

Taking this as the basic structure of our Node:



```
struct Node
{
int data;


Node *link;
};
```

Implementation using Linked List

As we know that we use a head pointer to keep track of the starting of our linked list, So when we are implementing stack using linked list we can simply call the head pointer as top to make it more relatable to stack.

Push (insert element in stack)

The push operation would be similar to inserting a node at starting of the linked list So initially when the Stack (Linked List) is empty, the top pointer will be NULL. Let's suppose we have to insert the values 1, 2 & 3 in the stack.
So firstly we will create a new Node using the new operator and return its address in temporary pointer ptr.
Then we will insert the value 1 in the data part of the Node : ptr->data = valueand make link part of the node equal to top : ptr->link=top.
Finally we will make top = ptr to point it to the newly created node which will now be the starting of the linked list and top of our stack.

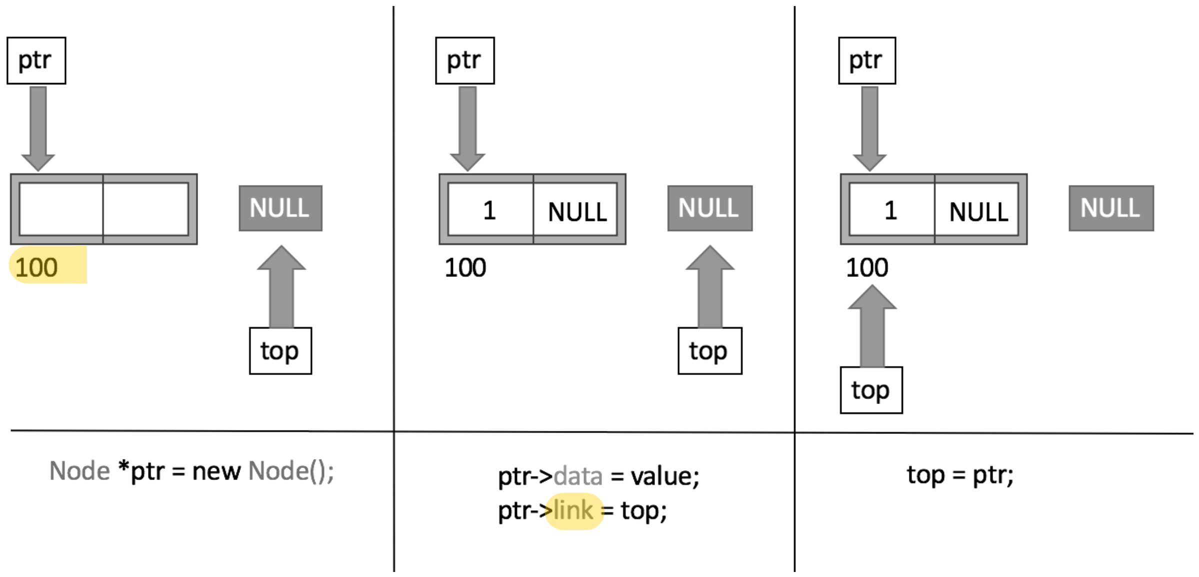


Figure 5: Push() method using linked list

Similarly we can push the values 2 & 3 in the stack which will give us a linked list of three nodes with top pointer pointing to the node containing value 3.

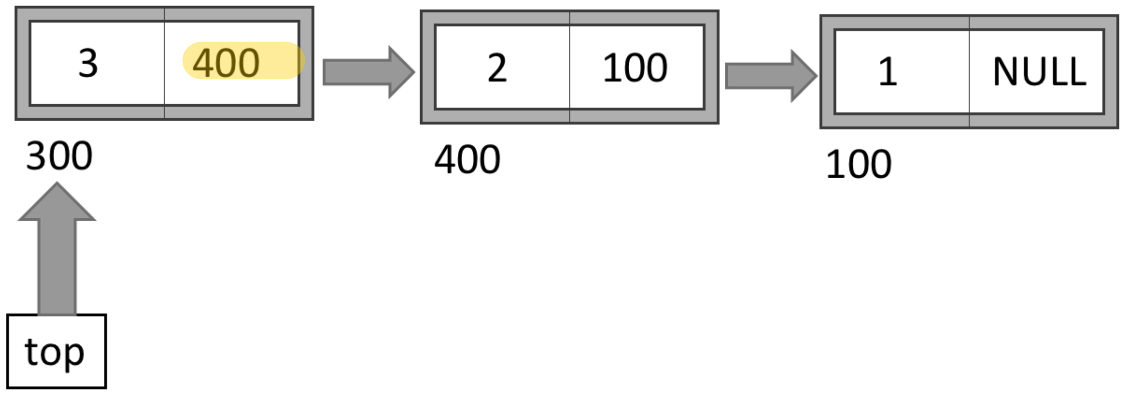


Figure 6: Adding the third value in the linked list using the Push() method.

Pop (delete element from stack)

1. The pop operation would be similar to deleting a node from the starting of a linked list.
2. So we will take a temporary pointer ptr and equate it to the top pointer.
3. Then we will move the top pointer to the next node i.e. top = top->link
4. Finally, we will delete the node using delete operator and pointer ptr i.e delete(ptr)

After we pop once our stack will look something like this:

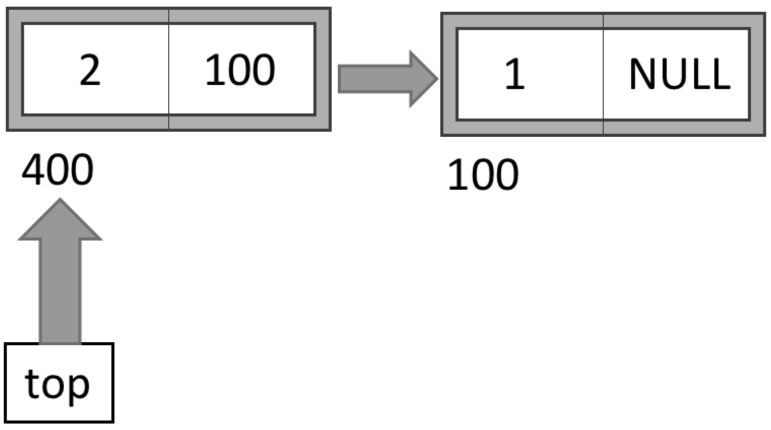


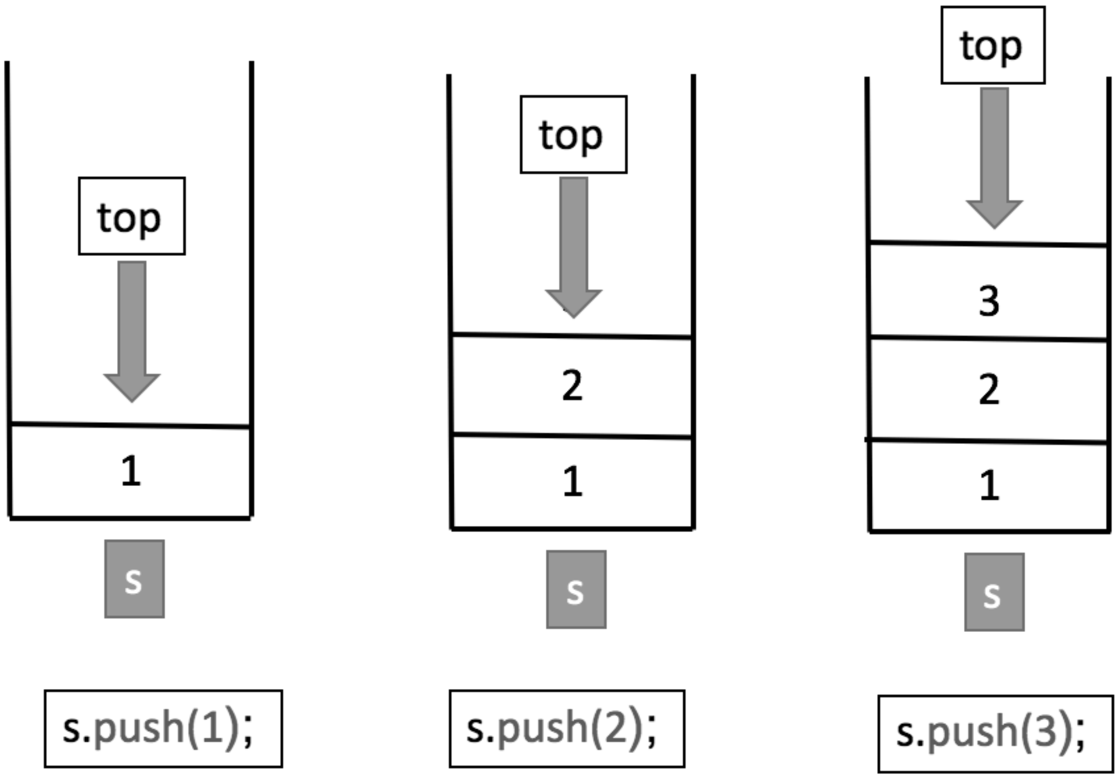
Figure 7: After popping the third value from the stack

IsEmpty (check if stack is empty or not)

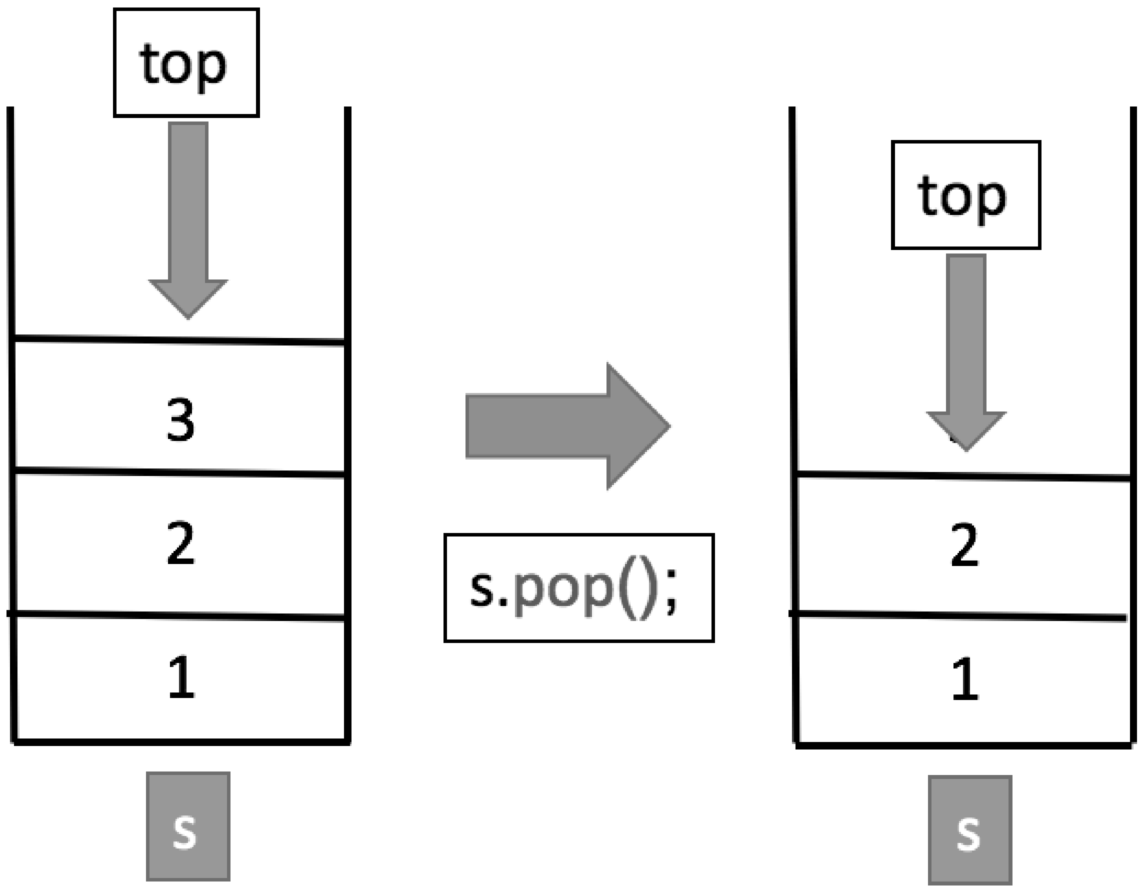
To check if the stack is empty or not we can simply check if top == NULL, it means that the stack is empty.

```
#include <iostream>
using namespace std;

//Structure of the Node
struct Node
{
int data;

Node *link;
};

// top pointer to keep track of the top of the stack
Node *top = NULL;

//Function to check if stack is empty or not
bool isempty()
{
 if(top == NULL)
 return true; else
 return false;
}

//Function to insert an element in stack
void push (int value)
{
  Node *ptr = new Node();
  ptr->data = value;
  ptr->link = top;
  top = ptr;
}

//Function to delete an element from the stack
void pop ( )
{
 if ( isempty() )
  cout<<"Stack is Empty";
 else
 {
  Node *ptr = top;
  top = top -> link;
  delete(ptr);
 }
}

// Function to show the element at the top of the stack
void showTop()
{
 if ( isempty() )
  cout<<"Stack is Empty";
 else
  cout<<"Element at top is : "<< top->data;
}

// Function to Display the stack
void displayStack()
{
 if ( isempty() )
  cout<<"Stack is Empty";
 else
 {
  Node *temp=top;
  while(temp!=NULL)
  {   cout<<temp->data<<" ";
   temp=temp->link;
  }
  cout<<"\n";
 }
 }

// Main function
int main()
{

 int choice, flag=1, value;

 //Menu Driven Program using Switch
 while( flag == 1)
 {
```

```
 cout<<"\n1.Push 2.Pop 3.showTop 4.displayStack 5.exit\n";
 cin>>choice;
 switch (choice)
 {
 case 1: cout<<"Enter Value:\n";
         cin>>value;
         push(value);
         break;
 case 2: pop();
         break;
 case 3: showTop();
         break;
 case 4: displayStack();
         break;
 case 5: flag = 0;
         break;
 }
 }

return 0;
}
```



pop - To delete an element from the top of the stack we have the the void pop() method. So if we want to remove the top most element from the stack we can simply do s.pop();



top - to return the element at the top of the stack we have the top() method.

cout<<s.top(); will produce the output 2.
empty - To check if the stack is empty or not we have the method bool empty() which returns 0 if the stack is not empty and returns 1, if the stack is empty.

for stack s cout<<s.empty(); will produce the output as 0 because the stack is not empty.
size - The size() method returns us the size of the stack that is the total number of elements in the stack.

cout<<s.size(); will give the output as 2.
top - to return the element at the top of the stack we have the top() method.

cout<<s.top(); will produce the output 2.
empty - To check if the stack is empty or not we have the method bool empty() which returns 0 if the stack is not empty and returns 1, if the stack is empty.

for stack s cout<<s.empty(); will produce the output as 0 because the stack is not empty.
size - The size() method returns us the size of the stack that is the total number of elements in the stack.

cout<<s.size(); will give the output as 2.

```
#include <iostream>
#include <stack>
using namespace std;

//function to display stack
void displayStack(stack<int> s)
{
 int n = s.size();

  for(int i=0; i<n;i++)
 {
  cout<<s.top()<<" ";
  s.pop();
 }
 cout<<"\n";

 }

// Main function
int main()
{
  stack<int> s;

   //push - inserting elements 1,2 & 3 in stack
  s.push(1);
  s.push(2);
  s.push(3);

   //display the stack
  cout<<"Elements of Stack are : ";
  displayStack(s);

   //pop - deleting the element at top
  s.pop();

   cout<<"Stack after pop operation is : ";
  displayStack(s);

   //Display element at top
  cout<<"Element at top is : "<<s.top()<<"\n";

   //to check if queue is empty or not
  cout<<"Stack is empty (1 - YES / 0 - NO) : "<<s.empty()<<"\n";

   //Size of stack
  cout<<"Size of stack is : "<<s.size()<<"\n";

   return 0;
}
```