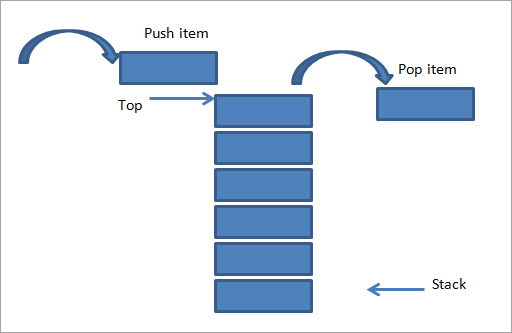
**Stack Data Structure In C++**

* Stack is a fundamental data structure which is used to store elements in a linear fashion.
* Stack follows **LIFO (last in, first out)** order or approach in which the operations are performed.
* This means that the element which was added last to the stack will be the first element to be removed from the stack.

## Stack in C++:-

A stack is similar to real-life stack or a pile of things that we stack one above the other.

**Given below is a pictorial representation of Stack.**



As shown above, there is a pile of plates stacked on top of each other. If we want to add another item to it, then we add it at the top of the stack as shown in the above figure (left-hand side). This operation of adding an item to stack is called “**Push**”.

On the right side, we have shown an opposite operation i.e. we remove an item from the stack. This is also done from the same end i.e. the top of the stack. This operation is called “**Pop**”.

As shown in the above figure, we see that push and pop are carried out from the same end. This makes the stack to follow LIFO order. The position or end from which the items are pushed in or popped out to/from the stack is called the “**Top of the stack**”.

Initially, when there are no items in the stack, the top of the stack is set to -1. When we add an item to the stack, the top of the stack is incremented by 1 indicating that the item is added. As opposed to this, the top of the stack is decremented by 1 when an item is popped out of the stack.

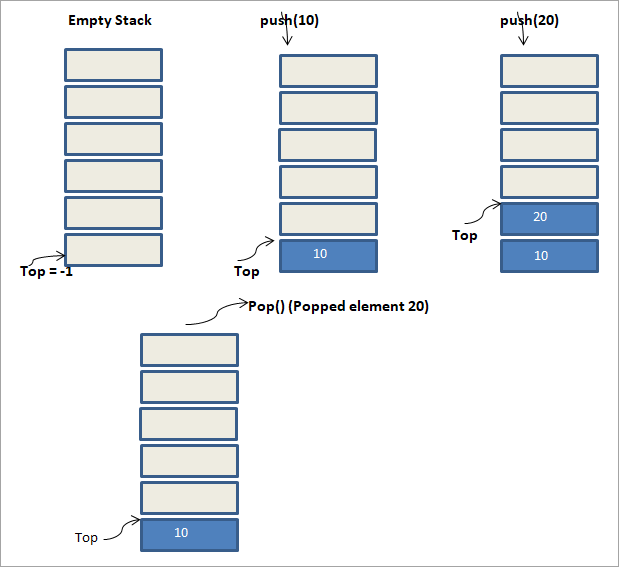
Next, we will see some of the basic operations of the stack data structure that we will require while implementing the stack.

### **Basic Operations**

Following are the basic operations that are supported by the stack.

* **push –** Adds or pushes an element into the stack.
* **pop –**Removes or pops an element out of the stack.
* **peek –** Gets the top element of the stack but doesn’t remove it.
* **isFull –**Tests if the stack is full.
* **isEmpty –** Tests if the stack is empty.

### **Illustration**



The above illustration shows the sequence of operations that are performed on the stack. Initially, the stack is empty. For an empty stack, the top of the stack is set to -1.

Next, we push the element 10 into the stack. We see that the top of the stack now points to element 10.

Next, we perform another push operation with element 20, as a result of which the top of the stack now points to 20. This state is the third figure.

Now in the last figure, we perform a pop () operation. As a result of the pop operation, the element pointed at the top of the stack is removed from the stack. Hence in the figure, we see that element 20 is removed from the stack. Thus the top of the stack now points to 10.

In this way, we can easily make out the LIFO approach used by stack.

### **Implementation**

#### **1.Write a C++ program to implement stack using arrays.**

#include <iostream>

using namespace std;

int **stack[100], n = 100, top = -1;**

void push(int val) push(10)20

{

   if(top >= n-1)-1>=99 0>=99 100>=99

      cout<<"Stack Overflow"<<endl;

   else

{

      top++; 0 1

      stack[top] = val;stack[0]=10 stack[1]=20

   }

}

void pop() {

   if(top <= -1)1<=-1 0<=-1 -1<=-1

      cout<<"Stack Underflow"<<endl;

   else {

      cout<<"The popped element is "<< stack[top] <<endl;stack[1]=20stack[0]=10

      top--;0 -1

   }

}

void display()

{

   if(top>= 0) 1>=0 0>=0

{

      cout<<"Stack elements are:";

      for(int i = top(1)(0); i>= 0; i--)

         cout<<stack[i]<<" ";20 10

      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) Display 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;10

            push(val);push(10)(20)

            break;

         }

         case 2: {

            pop();

            break;

         }

         case 3: {

            display();

            break;

         }

         case 4: {

            cout<<"Exit"<<endl;

            break;

         }

         default: {

            cout<<"Invalid Choice"<<endl;

         }

      }

   }while(ch! = 4);

return 0;

}