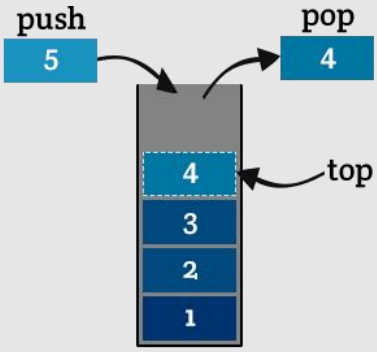
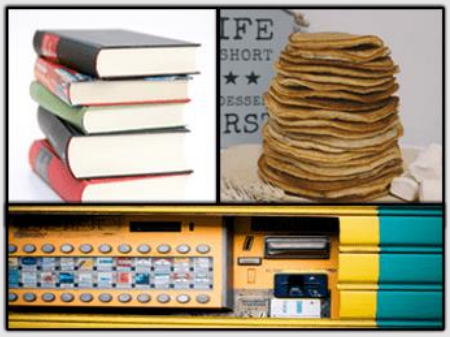
Lab 04
Stack Data Structure





Prepared by: Engr. Kiran

FCSE, GIKI

Lab# 04
Stack Data Structure in C++

4.1 Introduction to Stack in C++

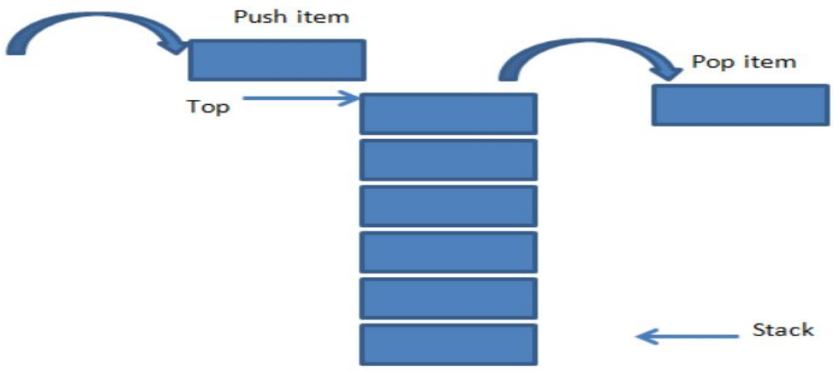
A stack is a linear data structure that contains a collection of elements in which the insertion of a new element and removal of an existing element takes place at the same end represented as the top of the stack.

To implement the stack, it is required to maintain the **pointer to the top of the stack**, which is the last element to be inserted because **we can access the elements only on the top of the stack.**

4.1.1 LIFO(Last In First Out):

Both insertion and removal are allowed at only one end of Stack called Top.The element that is inserted last will come out first. You can take a pile of plates kept on top of each other as a real-life example. The plate which we put last is on the top and since we remove the plate that is at the top, we can say that the plate that was put last comes out first.

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.



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.

4.2 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.

4.2.1 PUSH

This method allows us to add an element to the stack. Adding an element in a Stack in C++ occurs only at its top because of the LIFO policy.

In this process, the following steps are performed:

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.

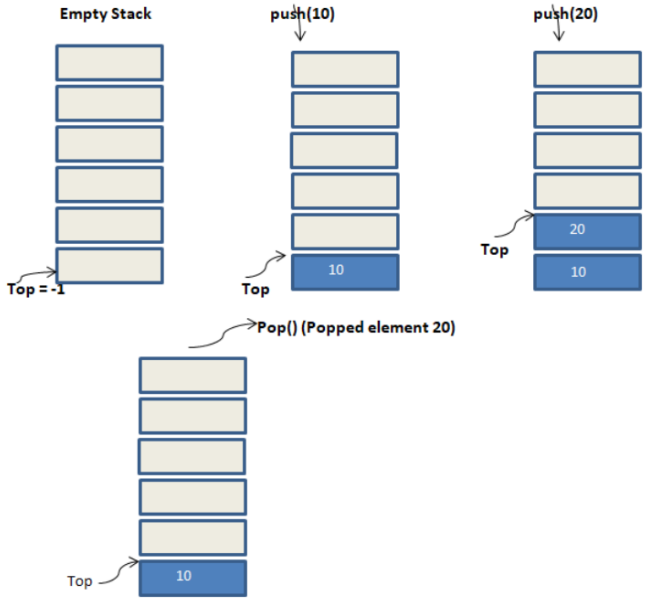
4.2.2 POP

This method allows us to remove an element from the top of the stack.

Dequeue operation consists of the following steps:

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.

4.2.3 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.

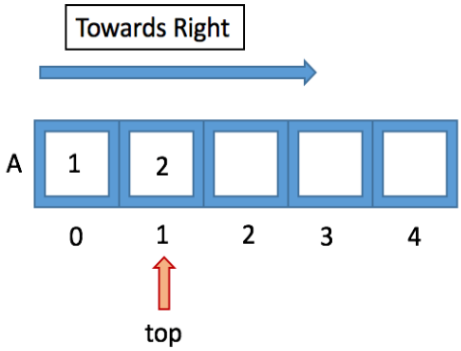
Next, we perform another push operation with element 20, as a result of which the top of the stack now points to 20. This state is the third figure.

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.

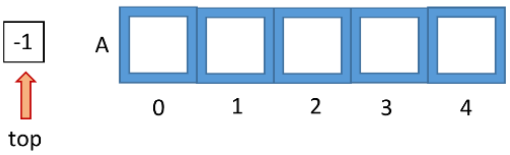
4.3 Array Implementation For Stack

Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size. Here we will implement Stack using array.

- When we implement stack uisng array we take the direction of the stack i.e the direction in which elements are inserted towards right.

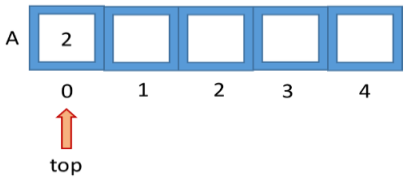


Lets take an example of an array of 5 elements to implement stack. So we define the size of the array using pre-processor directive `#deifne SIZE 5` & then we can create an array of 5 elements as `int A[SIZE];` Also we will declare a `top` variable to track the element at the top of the stack which will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

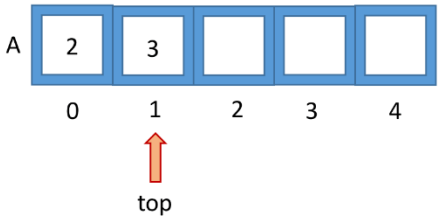


- `isempty()`- Now that we know that the stack is empty when `top` is equal to -1 we can use this to implement our `isempty()` function i.e. when `top == -1` retrun true else return false.
- `push()`- To push an element into the stack we will simply increment `top` by one and insert the element at that position. While inserting we have to take care of the condition when the array is full i.e. `when top == SIZE-1;`

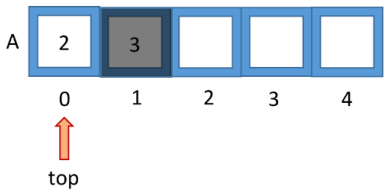
- `pop()`- To pop an element from the stack we will simple decrement `top` by one which will simply mean that the element is no longer the part of the stack. In this case we have to take care of the condition when the stack is empty i.e `top == -1` then we cannot perform the pop operation.
- `show_top()`-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



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



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



Position of Top	Status of Stack
-1	Stack is Empty
0	Only one element in Stack
N-1	Stack is Full
N	Overflow state of Stack

Let us implement the stack data structure using C++.

```
#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;
}
```

Advantages of array implementation:

- Easy to implement.
- Memory is saved as pointers are not involved.

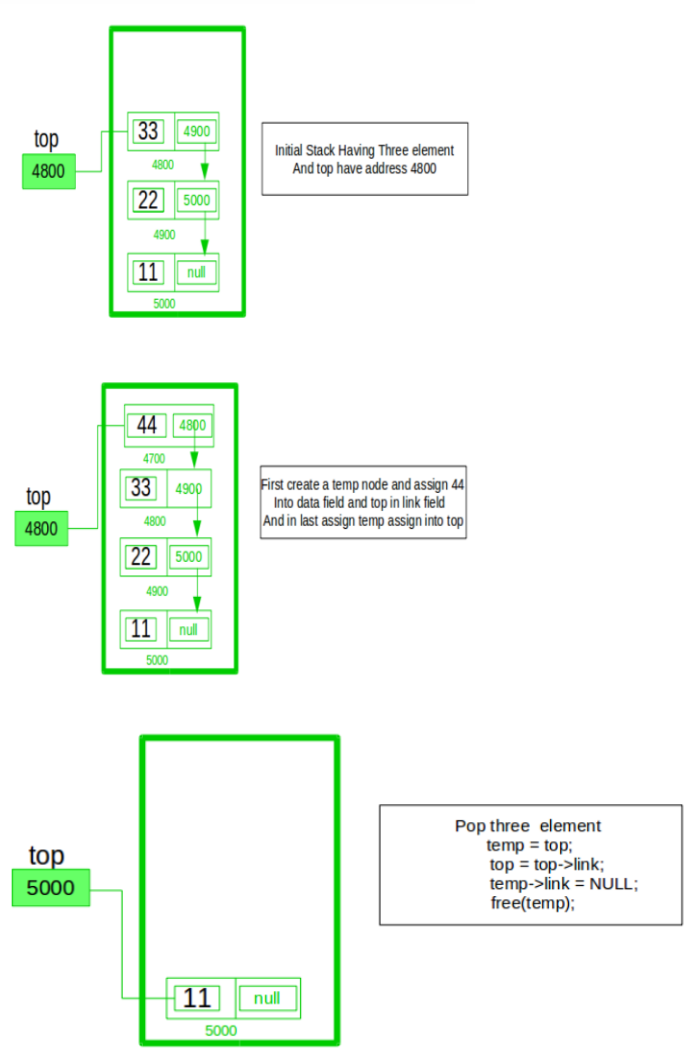
Disadvantages of array implementation:

- It is not dynamic i.e., it doesn't grow and shrink depending on needs at runtime. [But in case of dynamic sized arrays like vector in C++, list in Python, ArrayList in Java, stacks can grow and shrink with array implementation as well].
- The total size of the stack must be defined beforehand.

4.4 Linked List Implementation for Stack:

To implement a stack using the singly linked list concept, all the singly linked list operations should be performed based on Stack operations LIFO(last in first out) and with the help of that knowledge, we are going to implement a stack using a singly linked list.

So we need to follow a simple rule in the implementation of a stack which is **last in first out** and all the operations can be performed with the help of a top variable. Let us learn how to perform **Pop, Push, Peek, and Display** operations:



In the stack Implementation, a stack contains a top pointer. which is the “head” of the stack where pushing and popping items happens at the head of the list. The first node has a null in the link field and second node-link has the first node address in the link field and so on and the last node address is in the “top” pointer.

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  struct Node *next;
};
struct Node* top = NULL;
void push(int val) {
  struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
  newnode->data = val;
```

```
  newnode->next = top;
  top = newnode;
}
void pop() {
  if(top==NULL)
  cout<<"Stack Underflow"<<endl;
  else {
    cout<<"The popped element is "<< top->data <<endl;
    top = top->next;
  }
}
void display() {
  struct Node* ptr;
  if(top==NULL)
  cout<<"stack is empty";
  else {
    ptr = top;
    cout<<"Stack elements are: ";
    while (ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
    }
  }
  cout<<endl;
}
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;
        push(val);
        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;
}
```

Output:

```
1) Push in stack
2) Pop from stack
3) Display stack
4) Exit

Enter choice: 1
Enter value to be pushed: 2
Enter choice: 1
Enter value to be pushed: 6
```

```
Enter choice: 1
Enter value to be pushed: 8
Enter choice: 1
Enter value to be pushed: 7
Enter choice: 2
The popped element is 7
Enter choice: 3
Stack elements are:8 6 2
Enter choice: 5
Invalid Choice
Enter choice: 4
Exit
```

Advantages of Linked List implementation:

- The linked list implementation of a stack can grow and shrink according to the needs at runtime.
- It is used in many virtual machines like JVM.

Disadvantages of Linked List implementation:

- Requires extra memory due to the involvement of pointers.
- Random accessing is not possible in stack.

4.5 Analysis of Stack Operations

Below mentioned are the time complexities for various operations that can be performed on the Stack data structure.

Push Operation : O(1)

Pop Operation : O(1)

Top Operation : O(1)

Search Operation : O(n)

The time complexities for `push()` and `pop()` functions are `O(1)` because we always have to insert or remove the data from the **top** of the stack, which is a one step process.

4.6 Applications Of Stack

Let us discuss the various applications of the stack data structure below.

- CD/DVD stand.
- Stack of books in a book shop.
- Call center systems.
- Undo and Redo mechanism in text editors.