C Program to Implement Stack

```c
/* C program to implement stacks using linked list */
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
int pop (struct node **top)
        {
            int h;
            struct node *res = top;
            if (*top == NULL) {
               printf("underflow\n");
               return;
            }
            h = (*top)->data;
            *top = (*top)->next;
            free(res);
            return h;
        }
        struct node *create (int x)
        {
            struct node *newptr = (struct node
 *)malloc(sizeof(struct node));
            newptr->data = x;
            newptr->next = NULL;
            return newptr;
        }
        void push(struct node **top, int x)
        {
            struct node *newptr = create(x);
            if (*top == NULL)
            *top = newptr;
            else {
               newptr->next = (*top);
               (*top) = newptr;
            }
        }
        void main()
        {
            struct node *top = NULL;
            push(&top, 1);
            push(&top, 2);
            push(&top, 3);
            push(&top, 4);
            push(&top, 5);
            printf("%d\n", pop(&top));
            printf("%d\n", pop(&top));
        }
```

C++ Program to Implement Stack

```cpp
/*
 * C++ Program To Implement Stack using Linked List
 */
#include<iostream>
#include<cstdlib>
using namespace std;

/*
 * Node Declaration
         */
        struct node
        {
            int info;
            struct node *link;
        }*top;

        /*
         * Class Declaration
         */
        class stack_list
        {
            public:
                node *push(node *, int);
                node *pop(node *);
                void traverse(node *);
                stack_list()
                {
                    top = NULL;
                }
        };

        /*
         * Main: Contains Menu
         */
        int main()
        {
            int choice, item;
            stack_list sl;
            while (1)
            {
                cout<<"\n-------------"<<endl;
                cout<<"Operations on Stack"<<endl;
                cout<<"\n-------------"<<endl;
                cout<<"1.Push Element into the Stack"<<endl;
                cout<<"2.Pop Element from the Stack"<<endl;
                cout<<"3.Traverse the Stack"<<endl;
                cout<<"4.Quit"<<endl;
                cout<<"Enter your Choice: ";
                cin>>choice;
                switch(choice)
                {
                case 1:
                    cout<<"Enter value to be pushed into the stack: ";
                    cin>>item;
                    top = sl.push(top, item);
                    break;
                case 2:
                    top = sl.pop(top);
                    break;
                case 3:
                    sl.traverse(top);
                    break;
                case 4:
                    exit(1);
                    break;
                default:
                    cout<<"Wrong Choice"<<endl;
                }
            }
            return 0;
        }

        /*
         * Push Element into the Stack
         */
        node *stack_list::push(node *top, int item)
        {
            node *tmp;
            tmp = new (struct node);
            tmp->info = item;
            tmp->link = top;
            top = tmp;
            return top;
        }

        /*
         * Pop Element from the Stack
         */
        node *stack_list::pop(node *top)
        {
            node *tmp;
            if (top == NULL)
                cout<<"Stack is Empty"<<endl;
            else
            {
                tmp = top;
                cout<<"Element Popped: "<<tmp->info<<endl;
                top = top->link;
                delete(tmp);
            }
            return top;
        }

        /*
         * Traversing the Stack
         */
        void stack_list::traverse(node *top)
        {
            node *ptr;
            ptr = top;
            if (top == NULL)
                cout<<"Stack is empty"<<endl;
            else
            {
                cout<<"Stack elements :"<<endl;
                while (ptr != NULL)
                {
                    cout<<ptr->info<<endl;
                    ptr = ptr->link;
                }
            }
        }
```

C++ Program to Implement Stack in STL

```cpp
/*
 * C++ Program to Implement Stack in Stl
 */
#include <iostream>
#include <stack>
#include <string>
#include <cstdlib>
using namespace std;
int main()
        {
            stack<int> st;
            int choice, item;
            while (1)
            {
                cout<<"\n---------------------"<<endl;
                cout<<"Stack Implementation in Stl"<<endl;
                cout<<"\n---------------------"<<endl;
                cout<<"1.Insert Element into the Stack"<<endl;
                cout<<"2.Delete Element from the Stack"<<endl;
                cout<<"3.Size of the Stack"<<endl;
                cout<<"4.Top Element of the Stack"<<endl;
                cout<<"5.Exit"<<endl;
                cout<<"Enter your Choice: ";
                cin>>choice;
                switch(choice)
                {
                case 1:
                    cout<<"Enter value to be inserted: ";
                    cin>>item;
                    st.push(item);
                    break;
                case 2:
                    item = st.top();
                    st.pop();
                    cout<<"Element "<<item<<" Deleted"<<endl;
                    break;
                case 3:
                    cout<<"Size of the Queue: ";
                    cout<<st.size()<<endl;
                    break;
                case 4:
                    cout<<"Top Element of the Stack: ";
                    cout<<st.top()<<endl;
                    break;
                case 5:
                    exit(1);
                    break;
                default:
                    cout<<"Wrong Choice"<<endl;
                }
            }
            return 0;
        }
```

```
$ g++ stack.cpp
$ a.out
---------------------
```

Java - The Stack Class

Stack is a subclass of Vector that implements a standard last-in, first-out stack.

Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

```
Stack( )
```

Apart from the methods inherited from its parent class Vector, Stack defines the following methods

Sr.No.	Method & Description
1	**boolean empty()** Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements.
2	**Object peek( )** Returns the element on the top of the stack, but does not remove it.
3	**Object pop( )** Returns the element on the top of the stack, removing it in the process.
4	**Object push(Object element)** Pushes the element onto the stack. Element is also returned.
5	**int search(Object element)** Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, -1 is returned.

```
import java.util.*;
public class StackDemo {

   static void showpush(Stack st, int a) {
      st.push(new Integer(a));
      System.out.println("push(" + a + ")");
```

```java
      System.out.println("stack: " + st);
   }

   static void showpop(Stack st) {
      System.out.print("pop -> ");
      Integer a = (Integer) st.pop();
      System.out.println(a);
      System.out.println("stack: " + st);
   }

   public static void main(String args[]) {
      Stack st = new Stack();
      System.out.println("stack: " + st);
      showpush(st, 42);
      showpush(st, 66);
      showpush(st, 99);
      showpop(st);
      showpop(st);
      showpop(st);
      try {
         showpop(st);
      } catch (EmptyStackException e) {
         System.out.println("empty stack");
      }
   }
}
```

Python

Methods of Stack

Python provides the following methods that are commonly used with the stack.

- **empty() -** It returns true, it the stack is empty. The time complexity is O(1).
- **size() -** It returns the length of the stack. The time complexity is O(1).
- **top() -** This method returns an address of the last element of the stack. The time complexity is O(1).
- **push(g) -** This method adds the element 'g' at the end of the stack - The time complexity is O(1).
- **pop() -** This method removes the topmost element of the stack. The time complexity is O(1).

Implementation of Stack

Python offers various ways to implement the stack. In this section, we will discuss the implementation of the stack using Python and its module.

We can implement a stack in Python in the following ways.

- List
- dequeu
- LifeQueue

Implementation Using List

Python list can be used as the stack. It uses the **append()** method to insert elements to the list where stack uses the **push()** method. The list also provides the pop() method to remove the last element, but there are shortcomings in the list. The list becomes slow as it grows.

The list stores the new element in the next to other. If the list grows and out of a block of memory then Python allocates some memory. That's why the list become slow. Let's understand the following example –

```python
# initial empty stack
my_stack = []

# append() function to push
# element in the my_stack
my_stack.append('x')
my_stack.append('y')
my_stack.append('z')


print(my_stack)

# pop() function to pop
# element from my_stack in
# LIFO order
print('\nElements poped from my_stack:')
print(my_stack.pop())
print(my_stack.pop())
print(my_stack.pop())

print('\nmy_stack after elements are poped:')
print(my_stack)
```

```python
from collections import deque

my_stack = deque()

# append() function is used to push
# element in the my_stack
my_stack.append('a')
my_stack.append('b')
my_stack.append('c')

print('Initial my_stack:')
print(my_stack)

# pop() function is used to pop
# element from my_stack in
# LIFO order
print('\nElements poped from my_stack:')
print(my_stack.pop())
print(my_stack.pop())
print(my_stack.pop())

print('\nmy_stack after elements are poped:')
print(my_stack)
```

Output:

```
Initial my_stack:
deque(['a', 'b', 'c'])
Elements poped from my_stack:
c
b
a
my_stack after elements are poped:
deque([])
```

Implementation Using queue module

The queue module has the LIFO queue, which is the same as the stack. Generally, the queue uses **the put()** method to add the data and **the ()** method to take the data.

Below are a few methods that available in the queue.

- **empty() -** If queue empty, returns true; otherwise return false.
- **maxsize() -** This method is used to set the maximum number of elements allowed in the queue.
- **get() -** It returns and removes the element from the queue. Sometime. The queue can be empty; it waits until element is available.
- **full() -** It returns True if the queue is full. The queue is defined as maxsize = 0 by default. In this case, it will not return **True**.
- **put(item) -** It adds the element to the queue; if the queue is full, wait until space is available.
- **put_nowait(item) -** It adds the element into the queue without delaying.
- **qsize() -** It returns the size of the queue.

Let's understand the following example of the queue module.

Example -

```
# Implementing stack using the queue module
from queue import LifoQueue

# Initializing a my_stack stack with maxsize
my_stack = LifoQueue(maxsize = 5)

# qsize() display the number of elements
# in the my_stack
print(my_stack.qsize())

# put() function is used to push
# element in the my_stack
my_stack.put('x')
my_stack.put('y')
my_stack.put('z')
```

```python

print("Stack is Full: ", my_stack.full())
print("Size of Stack: ", my_stack.qsize())

# To pop the element we used get() function
# from my_stack in
# LIFO order
print('\nElements poped from the my_stack')
print(my_stack.get())
print(my_stack.get())
print(my_stack.get())

print("\nStack is Empty: ", my_stack.empty())
```

Output:

```
0
Stack is Full:  False
Size of Stack:  3

Elements poped from the my_stack
z
y
x
```