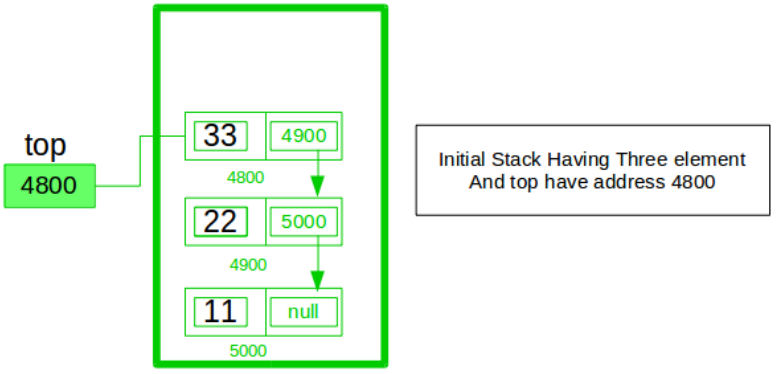
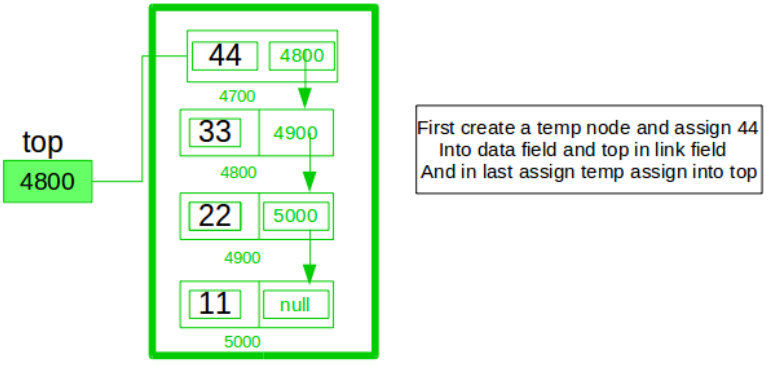
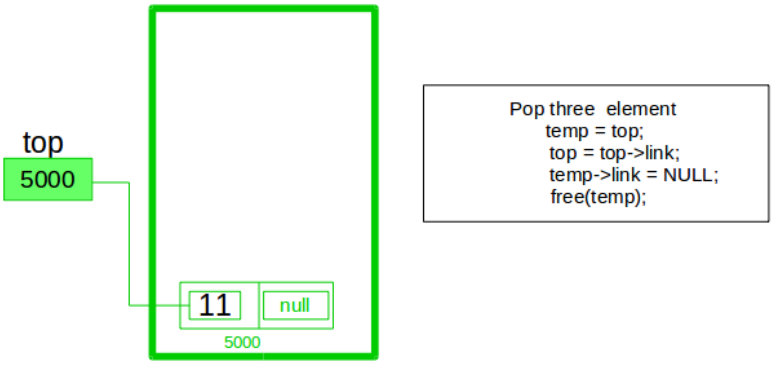
Linked List Implementation of Stack

To implement a stack using singly linked list concept , all the singly linked list operations are 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 single linked list. Using singly linked lists , we implement stack by storing the information in the form of nodes and we need to follow the stack rules and implement using singly linked list nodes . 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 ,Display** operations in the following article .







A stack can be easily implemented using the linked list. In stack Implementation, a stack contains a top pointer. which is "head" of the stack where pushing and popping items happens at the head of the list. First node have null in link field and second node link have first node address in link field and so on and last node address in "top" pointer.

The main advantage of using linked list over an arrays is that it is possible to implement a stack that can shrink or grow as much as needed. In using array will put a restriction to the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocate. so overflow is not possible.

Stack Operations:

1. **push( ) :** Insert a new element into stack i.e just inserting a new element at the beginning of the linked list.
2. **pop( ) :** Return top element of the Stack i.e simply deleting the first element from the linked list.
3. **peek( ):** Return the top element.
4. **display( ):** Print all elements in Stack.

Below is the implementation of the above approach:

```
// C++ program to Implement a stack
//using singly linked list
#include <bits/stdc++.h>
using namespace std;
```

```
// Declare linked list node

struct Node
{
    int data;
    Node* link;
};

Node* top;

// Utility function to add an element
// data in the stack insert at the beginning
void push(int data)
{

    // Create new node temp and allocate memory in heap
    Node* temp = new Node();

    // Check if stack (heap) is full.
    // Then inserting an element would
    // lead to stack overflow
    if (!temp)
    {
        cout << "\nStack Overflow";
        exit(1);
    }

    // Initialize data into temp data field
    temp->data = data;

    // Put top pointer reference into temp link
    temp->link = top;

    // Make temp as top of Stack
    top = temp;
}

// Utility function to check if
```

```
// the stack is empty or not
int isEmpty()
{
    //If top is NULL it means that
    //there are no elements are in stack
    return top == NULL;
}

// Utility function to return top element in a stack
int peek()
{

    // If stack is not empty , return the top element
    if (!isEmpty())
        return top->data;
    else
        exit(1);
}

// Utility function to pop top
// element from the stack
void pop()
{
    Node* temp;

    // Check for stack underflow
    if (top == NULL)
    {
        cout << "\nStack Underflow" << endl;
        exit(1);
    }
    else
    {

        // Assign top to temp
        temp = top;

        // Assign second node to top
```

```
        top = top->link;

        //This will automatically destroy
        //the link between first node and second node

        // Release memory of top node
        //i.e delete the node
        free(temp);
    }
}

// Function to print all the
// elements of the stack
void display()
{
    Node* temp;

    // Check for stack underflow
    if (top == NULL)
    {
        cout << "\nStack Underflow";
        exit(1);
    }
    else
    {
        temp = top;
        while (temp != NULL)
        {

            // Print node data
            cout << temp->data << "-> ";

            // Assign temp link to temp
            temp = temp->link;
        }
    }
}
```

```
// Driver Code
int main()
{

    // Push the elements of stack
    push(11);
    push(22);
    push(33);
    push(44);

    // Display stack elements
    display();

    // Print top element of stack
    cout << "\nTop element is "
         << peek() << endl;

    // Delete top elements of stack
    pop();
    pop();

    // Display stack elements
    display();

    // Print top element of stack
    cout << "\nTop element is "
         << peek() << endl;

    return 0;
}
```

Output:

```
44->33->22->11->
Top element is 44
22->11->
Top element is 22
```

Time Complexity:

The time complexity for all push(), pop(), and peek() operations is O(1) as we are not performing any kind of traversal over the list. We perform all the operations through the current pointer only.