جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم المن السيبراني

Department of Cyber Security

Subject: Data Structure

Class: Second

Lecturer: Msc :Muntather AL-mussawee

Lecture: (5)

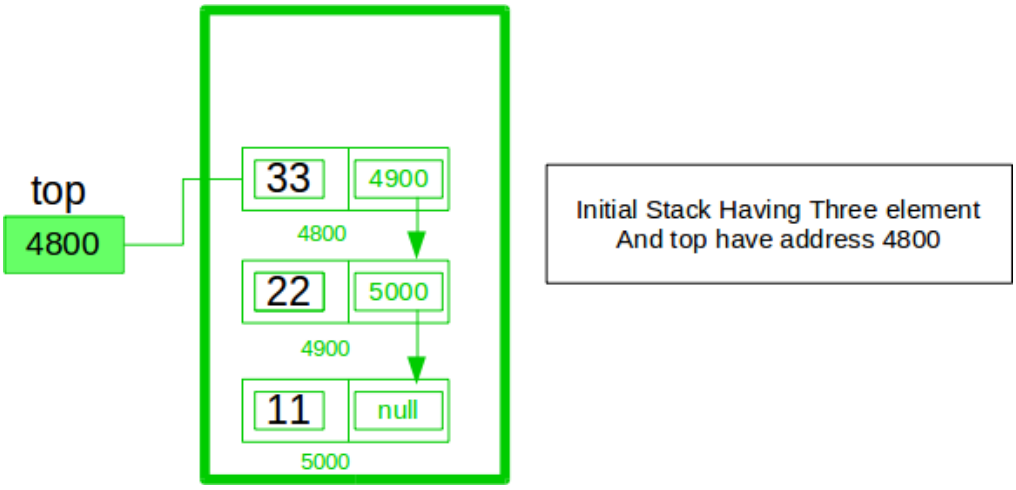
Implement a stack using singly linked list

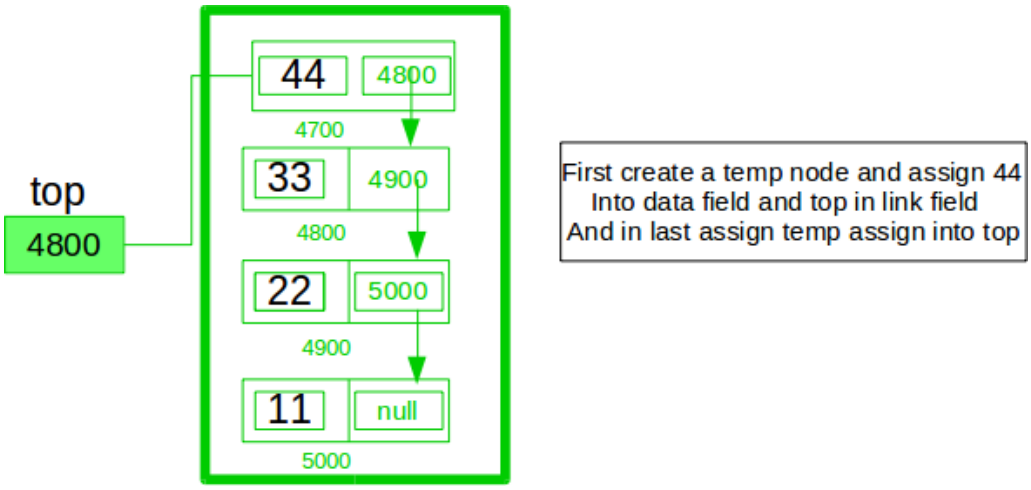
Study Year: 2025-2026

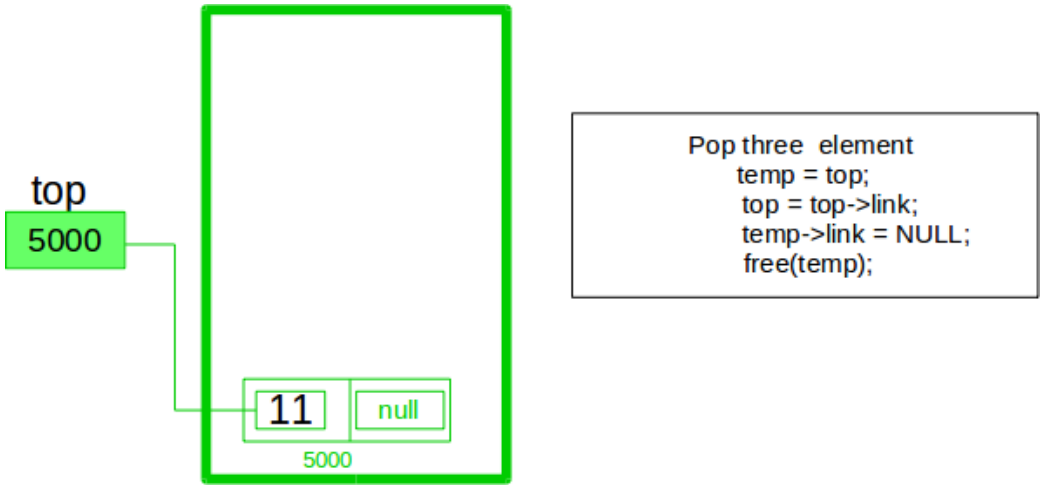
Implement a stack using singly linked list

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 following article:







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.

The main advantage of using a linked list over arrays is that it is possible to

array will put a restriction on the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocated. so overflow is not possible.

Stack Operations:

- **push( ) :** Insert a new element into the stack i.e just insert a new element at the beginning of the linked list.
- **pop( ) :** Return the top element of the Stack i.e simply delete the first element from the linked list.
- **peek( ) :** Return the top element.
- **display( ) :** Print all elements in Stack.

Push Operation:

- *Initialise a node*
- *Update the value of that node by data i.e.* ***node->data = data***
- *Now link this node to the top of the linked list*
- *And update top pointer to the current node*

: Pop Operation

- *First Check whether there is any node present in the linked list or not, if not then return*
- *Otherwise make pointer let say **temp** to the top node and move forward the top node by 1 step*
- *Now free this temp node*

Peek Operation:

- *Check if there is any node present or not, if not then return.*
- *Otherwise return the value of top node of the linked list*

Display Operation:

- *Take a **temp** node and initialize it with top pointer*
- *Now start traversing temp till it encounters NULL*
- *Simultaneously print the value of the temp node*

Below is the implementation of the above operations

```
#include <bits/stdc++.h>
using namespace std;

// Define a Node in the linked list
struct Node {
   int data;
   Node* next;
};

// Initialize the head of the stack as a nullptr
Node* head = nullptr;

// Function to check if the stack is empty
bool isEmpty() {
   return head == nullptr;
}

// Function to push an element onto the stack
void push(int new_data) {
   // Allocate memory for a new node
   Node* new_node = new Node();
   new_node->data = new_data;

   // Check if memory allocation for the new node failed
   if (!new_node) {
      cout << "\nStack Overflow" << endl;
      return;
   }

   // Link the new node to the current top node
   new_node->next = head;
```

```
    // Update the top to the new node
    head = new_node;
}

// Function to remove the top element from the stack
void pop() {
    // Check for stack underflow
    if (isEmpty()) {
        cout << "\nStack Underflow" << endl;
        return;
    }

    // Temporary variable to hold the current top node
    Node* temp = head;

    // Update the top to the next node
    head = head->next;

    // Free the memory of the old top node
    delete temp;
}

// Function to return the top element of the stack
int peek() {
    if (!isEmpty()) {
        return head->data;
    } else {
        cout << "\nStack is empty" << endl;
        return INT_MIN;
    }
```

```
}

// Driver function to test stack implementation
int main() {
    // Push elements onto the stack
    push(11);
    push(22);
    push(33);
    push(44);

    // Print top element of the stack
    cout << "Top element is " << peek() << endl;

    // Remove two elements from the top
    cout << "Removing two elements..." << endl;
    pop();
    pop();

    // Print top element of the stack
    cout << "Top element is " << peek() << endl;

    return 0;
}
```