STACK USING LINKED LIST

Bachelor of Technology
Computer Science and Engineering

Submitted By

ARKAPRATIM GHOSH
(13000121058)

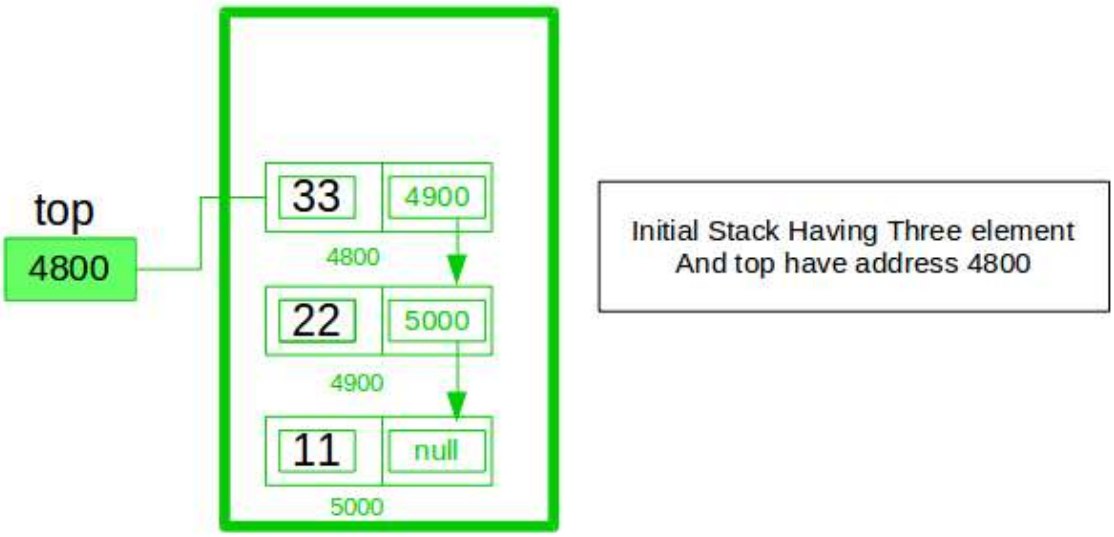
AUGUST 2022

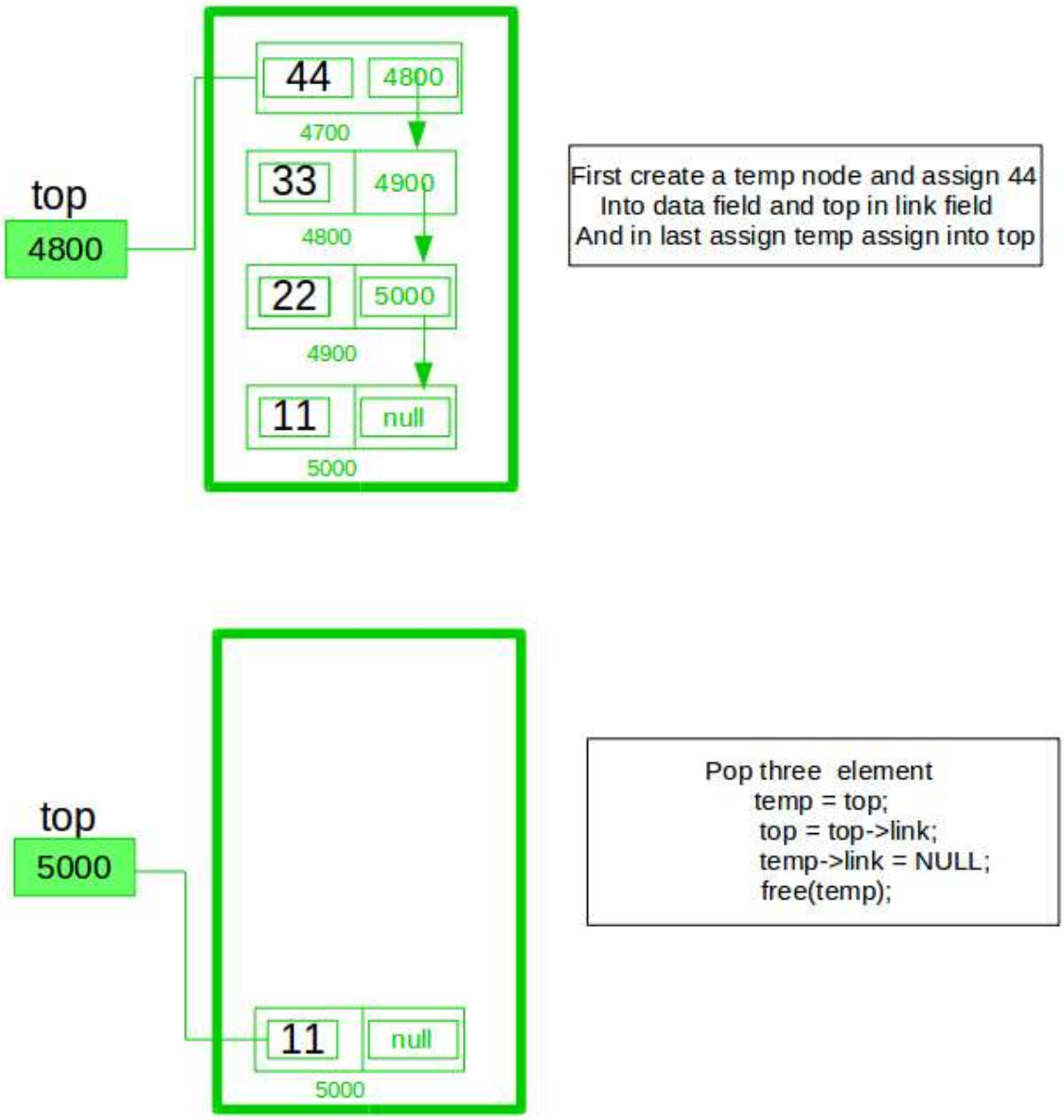
Techno Main
EM-4/1, Sector-V, Salt Lake
Kolkata- 700091
West Bengal
India

TABLE OF CONTENTS

INTRODUCTION

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 .





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.

5**.Size():**It returns the size of stack, i.e, the total number of items in a stack.

6**.isEmpty():**Returns a boolean value. It returns true if stack is empty else it return false.

A stack is represented using nodes of a linked list. Each node consists of two parts:data and next(storing address of next node). The data part of each node contains the assigned value and the next points to the node containing the next item in

the stack. The top refers to the topmost node in the stack. Both the push() and pop()operations are carried out at the front/top of the linked list and hence takes O(1)time.

PSEUDO CODE

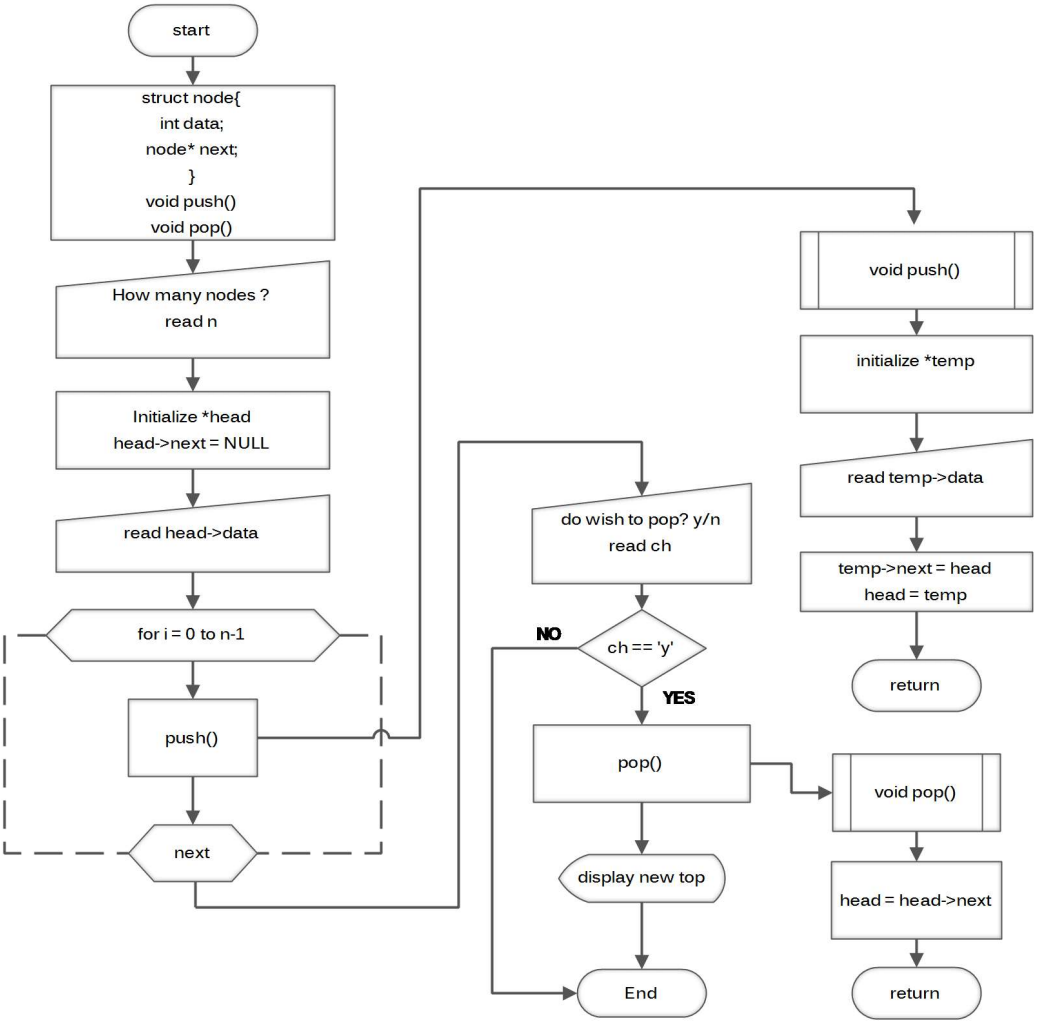
START

1.First we initialize the Node class which is the main element of out linked list, a node contains a variable to store data and a pointer to next node.
2.Now we create the class Stack.
3.We need a top pointer node for carrying out all operations related to the stack, it is initialized as NULL
4.Push() - function, is used to insert data into the stack by creation of a new node, so first we create a new node ' t ' in the heap if this operations fails that means the heap is full and so is the stack and we cannot push more data else add data into the new node ' t 'and make t->next =top and top point at t, so the top gets updated to the latest element inserted.
5.Pop() - function, is used to remove data by deleting a node from the linked list, so first we check if top == NULL i.e stack is empty and nothing to delete else get the data value in the top pointer and then return/print the data value and finally delte the pointer and update the top value to next available node.

6.Display() - function, is used to print all the data in the nodes and we iterated through all the nodes from the beginnning till we reach the last node i.e node==NULL.

END

FLOWCHART



EXPLANATION

How to push() elements in stack using linked list in C

Adding or inserting a new element to a stack is known asPush() operation in stack. Elements can only be pushed at top of the stack.

Steps to push an element into a Stack:

- Create a new node using dynamic memory allocation and assign value to the node.

```
struct Node *newNode = (struct
Node*)malloc(sizeof(struct Node));
newNode->data = 10;
```

- Check if stack is empty or not, i.e. (top == NULL).
- If is is empty, then set the next pointer of the node to NULL.

```
newNode->next = NULL;
```

- If it is not empty, the newly created node should be linked to the current top element of the stack, i.e,

```
newNode->next = top;
```

- Make sure that the top of the stack should always be pointing to the newly created node.

```
top = newNode;
```

Algorithm for push()

```
if top is equal to NULL
  newNode -> next = NULL
else
  newNode -> next = top
```

Example of Push() Operation:

```
// Structure to create a node with data and
next pointer

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

Node* top = NULL;

int pop() {
    if (top == NULL) {
        printf("\nEMPTY STACK");
    } else {
```

```
        struct Node *temp = top;
        int temp_data = top->data; //to store
data of top node
        top = top->next;
        free(temp); //deleting the node
        return temp_data;
    }
}
```

How to pop() elements from stack using linked list in C

Removing or deleting an element from a stack is known as Pop() operation in stack. Elements are popped from top of the stack. There should be at least one element in stack to perform pop() operation.

Steps to pop an element from a Stack:

- Check if stack is empty or not, i.e,(TOP == NULL).
- If it is empty, then print Stack Underflow.

```
print "Stack Underflow"
```

- If it is not empty, then create a temporary node and set it to top. Now, create another variable and copy the data of top element to this variable.

```
struct Node *temp = top;
int temp_data = top->data;
```

- Now, make top point to the next node.

```
top = top->next;
```

- Delete the temporary node and return the value stored in temp_data.

```
free(temp);
return temp_data;
```

Algorithm for pop()

```
if top == NULL
     print "Stack Underflow"
else
    create temporary node, *temp = top
    create temporary variable, temp_data = top-
>data
    top = top->next
    free(temp)
```

```
return temp_data
```

Example of Pop() Operation:

```
//Structure to create a node with data and next
pointer

struct Node {
    int data;
    struct Node *next;
}
Node* top = NULL;

int pop() {
    if (top == NULL) {
        printf("\nEMPTY STACK");
    } else {
        struct Node *temp = top;
        int temp_data = top->data; //to store
data of top node
        top = top->next;
        free(temp); //deleting the node
        return temp_data;
    }
}
```

Program to implement Stack using Linked List in C language

```c
#include <stdio.h>
#include <stdlib.h>

// Structure to create a node with data and next pointer
struct Node {
    int data;
    struct Node *next;
};
Node* top = NULL;

// Push() operation on a  stack
void push(int value) {
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value; // assign value to the node
    if (top == NULL) {
        newNode->next = NULL;
    } else {
        newNode->next = top; // Make the node as top
```

```
    }
    top = newNode; // top always points to the
newly created node
    printf("Node is Inserted\n\n");
}

int pop() {
    if (top == NULL) {
        printf("\nStack Underflow\n");
    } else {
        struct Node *temp = top;
        int temp_data = top->data;
        top = top->next;
        free(temp);
        return temp_data;
    }
}

void display() {
    // Display the elements of the stack
    if (top == NULL) {
        printf("\nStack Underflow\n");
    } else {
        printf("The stack is \n");
        struct Node *temp = top;
        while (temp->next != NULL) {
```

```
            printf("%d--->", temp->data);
            temp = temp->next;
        }
        printf("%d--->NULL\n\n", temp->data);
    }
}

int main() {
    int choice, value;
    printf("\nImplementaion of Stack using
Linked List\n");
    while (1) {
        printf("1. Push\n2. Pop\n3. Display\n4.
Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("\nEnter the value to
insert: ");
            scanf("%d", &value);
            push(value);
            break;
        case 2:
            printf("Popped element is :%d\n",
pop());
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("\nWrong Choice\n");
        }
    }
}
```

Output:

Push Operation:

```
Implementaion of Stack using Linked List
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 12
Node is Inserted
```

```
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 45
Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 56
Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3
```

```
The stack is
56--->45--->12--->NULL
```

Pop Operation:

```
The stack is
56--->45--->12--->NULL

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2
Popped element is :56
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2
Popped element is :45
1. Push
2. Pop
3. Display
4. Exit
```

```
Enter your choice : 3
The stack is
12--->NULL

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2
Popped element is :12
1. Push
2. Pop
3. Display
4. Exit
```

CONCLUSION

Stack is linear data structure which follows theLast in, First Out Principle (LIFO).

- Stack can be represented using nodes of a linked list.
- Stack supports operations such as push, pop, size, peek and is Empty.
- Elements can be pushed or popped from one end only.
- Push and Pop operations take O(1) time.

REFERENCE

The information in this project report has been taken from the following:

Book:

DATA STRUCTURES THROUGH C IN DEPTH by S.K. SRIVASTAVA, DEEPALI SRIVASTAVA.

Links:

1.Stack Using Linked List in C - Scaler Topics

2.*Explain the stack by using linked list in C language (tutorialspoint.com)*