



Sri
SAI RAM
ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY

West Tambaram, Chennai - 44

Sairam
INSTITUTIONS



YEAR

2

SEM

3

CS8391

DATA STRUCTURES
(Common to CSE & IT)

UNIT No. 2

LINEAR STRUCTURES - STACKS, QUEUES

- STACK ADT
- OPERATIONS

Version: 1.XX



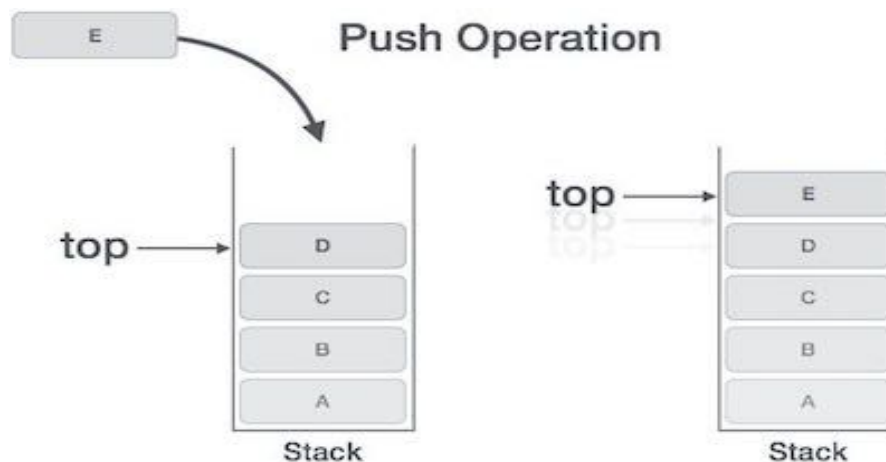
Stacks

Definition

A stack is linear data structures, a container of elements that are inserted and removed according to the last-in first-out (LIFO) principle. A stack is an ordered list of elements of the same data type.

for example – a deck of cards or a pile of plates, etc.

Stack Representation



Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations –

- push() – Pushing (storing) an element on the stack.
- pop() – Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks –

- peek() – get the top data element of the stack, without removing it.
- isFull() – check if stack is full.
- isEmpty() – check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named top. The top pointer provides the top value of the stack without actually removing it.

Stack Using Array

A stack data structure can be implemented using a one-dimensional array. But stack implemented using array stores only a fixed number of data values. This implementation is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable called 'top'. Initially, the top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

Stack Operations using Array

A stack can be implemented using an array as follows...

Before implementing actual operations, first follow the below steps to create an empty stack.

Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2 - Declare all the functions used in stack implementation.

Step 3 - Create a one dimensional array with fixed size (int stack[SIZE])

Step 4 - Define a integer variable 'top' and initialize with '-1'. (int top = -1)

Step 5 - In the main method, display a menu with a list of operations and make suitable function calls to perform operations selected by the user on the stack.

push(value) - Inserting value into the stack

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as a parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

Step 1 - Check whether stack is FULL. ($\text{top} == \text{SIZE}-1$)

Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.

Step 3 - If it is NOT FULL, then increment top value by one ($\text{top}++$) and set $\text{stack}[\text{top}]$ to value ($\text{stack}[\text{top}] = \text{value}$).

pop() - Delete a value from the Stack

In a stack, $\text{pop}()$ is a function used to delete an element from the stack. In a stack, the element is always deleted from the top position. Pop function does not take any value as a parameter. We can use the following steps to pop an element from the stack...

Step 1 - Check whether stack is EMPTY. ($\text{top} == -1$)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then delete $\text{stack}[\text{top}]$ and decrement top value by one ($\text{top}--$).

display() - Displays the elements of a Stack

We can use the following steps to display the elements of a stack...

Step 1 - Check whether stack is EMPTY. ($\text{top} == -1$)

Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize it with top. Display $\text{stack}[\text{i}]$ value and decrement i value by one ($\text{i}--$).

Step 3 - Repeat above step until i value becomes '0'.

Implementation of Stack using Array

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

```
#include<conio.h>
```

```
#define SIZE 10

void push(int);

void pop();

void display();

int stack[SIZE], top = -1;

void main()

{

    int value, choice;

    clrscr();

    while(1){

        printf("\n\n***** MENU *****\n");

        printf("1. Push\n2. Pop\n3. Display\n4. Exit");

        printf("\nEnter your choice: ");

        scanf("%d",&choice);

        switch(choice){

            case 1: printf("Enter the value to be insert: ");

                    scanf("%d",&value);

                    push(value);

                    break;

            case 2: pop();

                    break;

            case 3: display();
```

```
        break;

    case 4: exit(0);

    default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void push(int value){

    if(top == SIZE-1)

        printf("\nStack is Full!!! Insertion is not possible!!!");

    else{

        top++;

        stack[top] = value;

        printf("\n Insertion success!!!");

    }

}

void pop(){

    if(top == -1)

        printf("\nStack is Empty!!! Deletion is not possible!!!");

    else{

        printf("\nDeleted : %d", stack[top]);

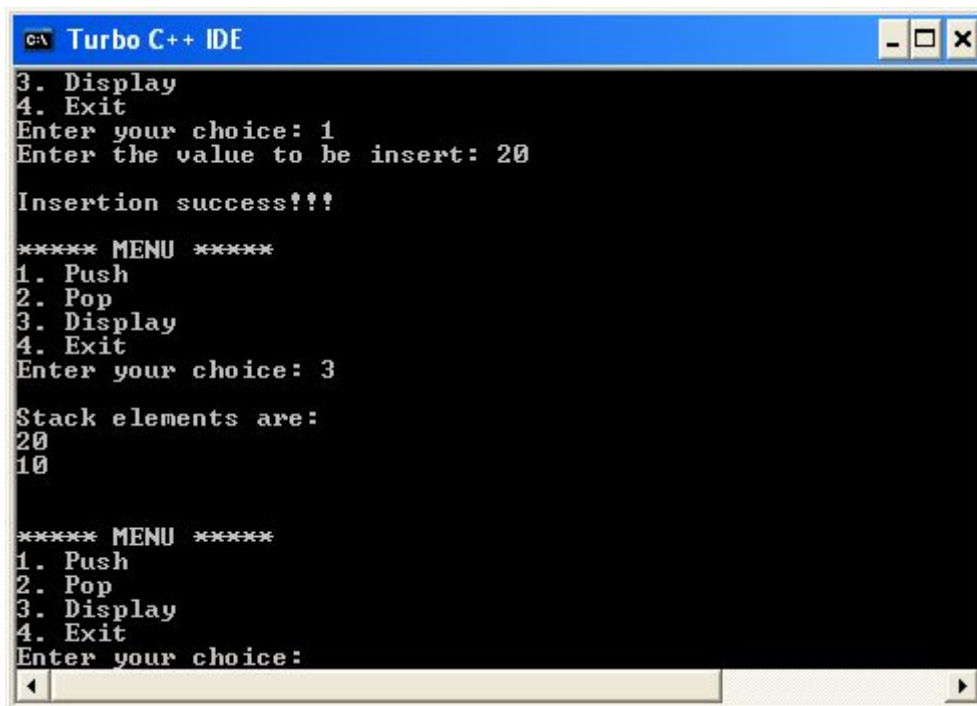
        top--;

    }

}
```

```
}  
  
void display(){  
    if(top == -1)  
        printf("\nStack is Empty!!!");  
    else{  
        int i;  
        printf("\nStack elements are:\n");  
        for(i=top; i>=0; i--)  
            printf("%d\n",stack[i]);  
    }  
}
```

Output



```
Turbo C++ IDE  
3. Display  
4. Exit  
Enter your choice: 1  
Enter the value to be insert: 20  
Insertion success!!!  
***** MENU *****  
1. Push  
2. Pop  
3. Display  
4. Exit  
Enter your choice: 3  
Stack elements are:  
20  
10  
***** MENU *****  
1. Push  
2. Pop  
3. Display  
4. Exit  
Enter your choice:
```

Stack Using Linked List

The major problem with the stack implemented using an array is, it works only for a fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using an array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using a linked list data structure. The stack implemented using linked lists can work for an unlimited number of values. That means, a stack implemented using linked lists works for the variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its previous node in the list. The next field of the first element must be always NULL.

Example



In the above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32, 50 and 99.

Stack Operations using Linked List

To implement a stack using a linked list, we need to set the following things before implementing actual operations.

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.

Step 2 - Define a 'Node' structure with two members data and next.

Step 3 - Define a Node pointer 'top' and set it to NULL.

Step 4 - Implement the main method by displaying the Menu with a list of operations and make suitable function calls in the main method.

push(value) - Inserting an element into the Stack

We can use the following steps to insert a new node into the stack...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether stack is Empty (top == NULL)
- Step 3 - If it is Empty, then set newNode → next = NULL.
- Step 4 - If it is Not Empty, then set newNode → next = top.
- Step 5 - Finally, set top = newNode.

pop() - Deleting an Element from a Stack

We can use the following steps to delete a node from the stack...

- Step 1 - Check whether stack is Empty (top == NULL).
- Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
- Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
- Step 4 - Then set 'top = top → next'.
- Step 5 - Finally, delete 'temp'. (free(temp)).

display() - Displaying stack of elements

We can use the following steps to display the elements (nodes) of a stack...

- Step 1 - Check whether stack is Empty ($\text{top} == \text{NULL}$).
- Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
- Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
- Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. ($\text{temp} \rightarrow \text{next} != \text{NULL}$).
- Step 5 - Finally! Display 'temp → data ---> NULL'.

Implementation of Stack using Linked List | C Programming

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

```
#include<conio.h>
```

```
struct Node
```

```
{
```

```
    int data;
```

```
    struct Node *next;
```

```
}*top = NULL;
```

```
void push(int);
```

```
void pop();
```

```
void display();
```

```
void main()
```

```
{
```

```
    int choice, value;
```

```
    clrscr();
```

```
printf("\n:: Stack using Linked List ::\n");

while(1){

    printf("\n***** MENU *****\n");

    printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");

    printf("Enter your choice: ");

    scanf("%d",&choice);

    switch(choice){

        case 1: printf("Enter the value to be insert: ");

                scanf("%d", &value);

                push(value);

                break;

        case 2: pop(); break;

        case 3: display(); break;

        case 4: exit(0);

        default: printf("\nWrong selection!!! Please try again!!!\n");

    }

}

}

void push(int value)

{

    struct Node *newNode;

    newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
newNode->data = value;

if(top == NULL)

    newNode->next = NULL;

else

    newNode->next = top;

top = newNode;

printf("\nInsertion is Success!!!\n");

}

void pop()

{

    if(top == NULL)

        printf("\nStack is Empty!!!\n");

    else{

        struct Node *temp = top;

        printf("\nDeleted element: %d", temp->data);

        top = temp->next;

        free(temp);

    }

}

void display()

{

    if(top == NULL)
```

```
printf("\nStack is Empty!!!\n");

else{

    struct Node *temp = top;

    while(temp->next != NULL){

        printf("%d--->",temp->data);

        temp = temp -> next;

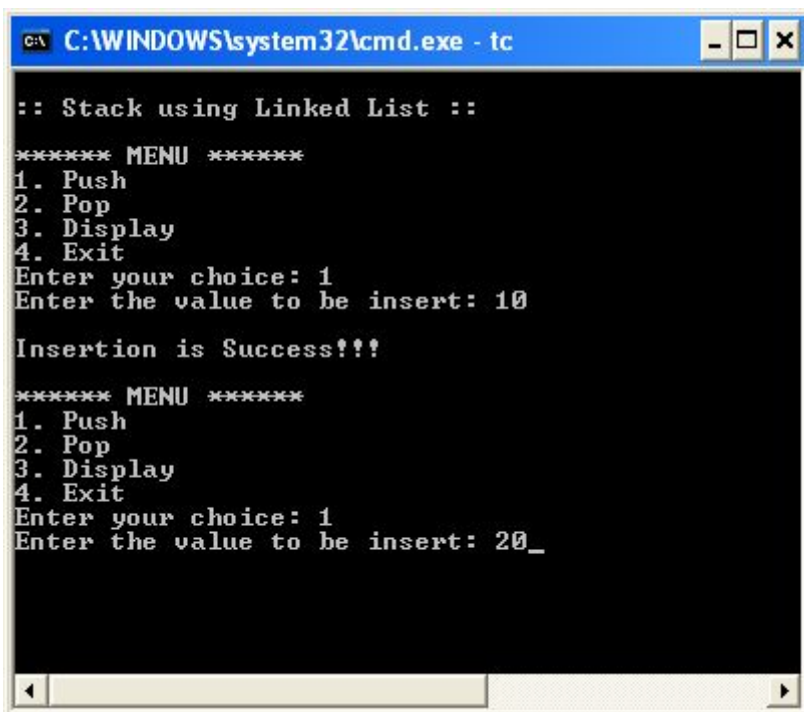
    }

    printf("%d--->NULL",temp->data);

}

}
```

Output



```
C:\WINDOWS\system32\cmd.exe - tc

:: Stack using Linked List ::

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 10

Insertion is Success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 20_
```

```
C:\WINDOWS\system32\cmd.exe - tc
4. Exit
Enter your choice: 2
Deleted element: 20
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
10--->NULL
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Deleted element: 10
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
```