

STACK ADT

Stack is an abstract data type and it is also called linear data structure. It follows last in, first out (LIFO) strategy. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO.

The stack operations are given below.

- Stack() - creates a new stack that is empty. push(item) - adds a new item to the top of the stack. pop ()- removes the top item from the stack.
- peek() - returns the top item from the stack but does not remove it.
- isEmpty() - tests to see whether the stack is empty. It returns a boolean value.
- size() - returns the number of items on the stack.

Stack using Array:

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 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. ($top == SIZE-1$)

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 ($top++$) and set $stack[top]$ to value ($stack[top] = value$).

pop() - Delete a value from the Stack

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

Step 1: Check whether stack is EMPTY. (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 stack[top] and decrement top value by one (top--).

display() - Displays the elements of a Stack

To display the elements of a stack

Step 1: Check whether stack is EMPTY. (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 with top. Display stack[i] value and decrement i value by one (i--).

Step 4: Repeat above step until i value becomes '0'.

Program:

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

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

```
#define SIZE 10
```

```
void push(int); void pop(); void display();
```

```
int stack[SIZE], top = -1;
```

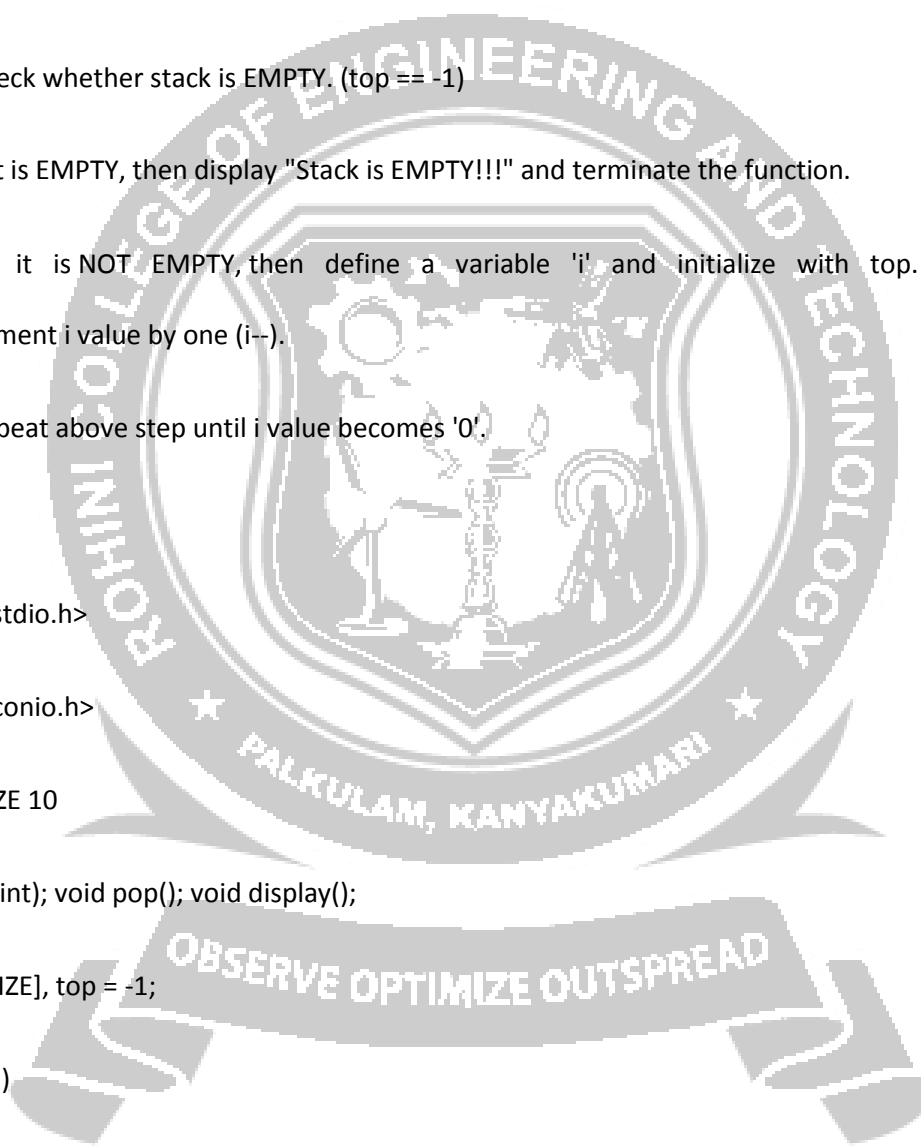
```
void main()
```

```
{
```


```
int value, choice;
```

```
clrscr();
```

```
while(1)
```



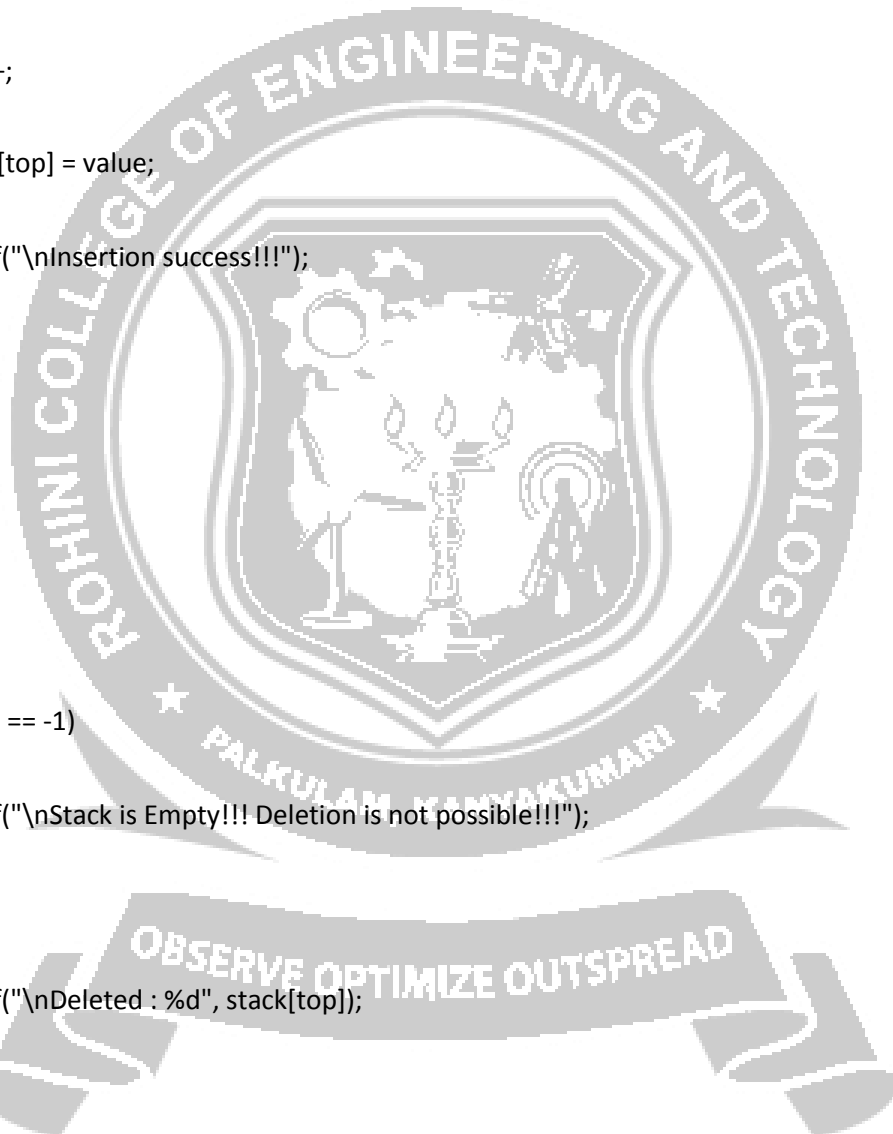
```
{  
  
printf("\n\n***** MENU *****\n");  
  
printf("1. Push\n2. Pop\n3. Display\n4. Exit"); printf("\nEnter your choice: "); scanf("%d",&choice);  
  
switch(choice){  
  
case 1:  
  
    printf("Enter the value to be insert: ");  
  
    scanf("%d",&value);  
  
    push(value);  
  
    break;  
  
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("\nInsertion 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{

inti;

printf("\nStack elements are:\n");

for(i=top; i>=0; i--)

printf("%d\n",stack[i]);

}

}

```

Stack using Linked List

- The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself.
- Stack implemented using 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 linked list data structure.
- The stack implemented using linked list can work for unlimited number of values. That means, stack implemented using linked list works for 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 next node in the list.
- The next field of the first element must be always NULL.

OPERATIONS

Step 1: Define a 'Node' structure with two member's data and next.

Step 2: Define a Node pointer 'top' and set it to NULL.

Step 3: Implement the main function by displaying Menu with list of operations and make suitable function calls in the main function.

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 \rightarrow next = NULL.

Step 4: If it is Not Empty, then set newNode \rightarrow 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 \rightarrow next'.

Step 7: 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 ($top == NULL$).

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 (temp → next != NULL).

Step 5: Finally, Display 'temp → data ---> NULL'.

Program:

```
#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);  
    }  
}
```

