Operations on a Stack

The following operations are performed on the stack...

1. Push (To insert an element on to the stack)
2. Pop (To delete an element from the stack)
3. Display (To display elements of the stack)

Stack data structure can be implemented in two ways. They are as follows...

1. Using Arrays
2. Using Linked List

Stack Using Arrays

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

We can Perform the following Operations on Stack

1.Push()
2.Pop()
3.Display()

A stack can be implemented using 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 main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

1.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**).

2.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--**).

3.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**. (**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'.

Source Code: To implement Stack Using Arrays

```
#include<iostream.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){
    cout<<"\n\n***** MENU *****\n";
    cout<<"1. Push\n2. Pop\n3. Display\n4. Exit";
    cout<<"\nEnter your choice: ";
    cin>>choice;
    switch(choice)
   {
          case 1: cout<<"Enter the value to be insert: ";
                  cin>>value;
                  push(value);
                  break;
         case 2: pop();
                  break;
```

```
            case 3: display();
                    break;
            case 4: exit(0);
            default: cout<<"\nWrong selection!!! Try again!!!";
      }
   }
}
void push(int value)
{
        if(top == SIZE-1)
       cout<<"\nStack is Full!!! Insertion is not possible!!!";
      else
       {
            top++;
            stack[top] = value;
             cout<<"\nInsertion success!!!";
       }
}
void pop()
{
            if(top == -1)
            cout<<"\nStack is Empty!!! Deletion is not possible!!!";
```

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

Output:

Signature of the Faculty

(ii) Stack Using Linked List

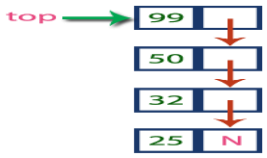
The major problem with the stack implemented using an arrays is, it works only for a fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself.

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 list can work for an unlimited number of values. That means, stack implemented using linked list 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

We can Perform the Following Operations on Stack Using Linked List (i.e)

1. Push()
2. Pop()
3. Display()

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 Menu with list of operations and

make suitable function calls in the **main**method.

1.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**.

2.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)**).

3.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**'.

```
#include<iostream.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();
    cout<<"\n:: Stack using Linked List ::\n";
    while(1)
    {
          cout<<"\n****** MENU ******\n";
          cout<<"1. Push\n2. Pop\n3. Display\n4. Exit\n";
           cout<<"Enter your choice: ";
           cin>>choice;
           switch(choice)
           {
               case 1:    cout<<"Enter the value to be insert: ";
                          cin>>value;
                          push(value);
                          break;
               case 2:   pop();
                         break;
               case 3:  display(); break;
               case 4: exit(0);
              default: cout<<"\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;
     cout<<"\nInsertion is Success!!!\n";
}
void pop()
{
       if(top == NULL)
       cout<<"\nStack is Empty!!!\n";
        else
        {
           struct Node *temp = top;
          cout<<"\nDeleted element:", temp->data;
          top = temp->next;
          free(temp);
       }
}
void display()
{
           if(top == NULL)
            cout<<"\nStack is Empty!!!\n";
        else
          {
             struct Node *temp = top;
             while(temp->next != NULL)
```

```
                cout<<temp->data;
                 temp = temp -> next;
            }
           cout<<temp->data;
        }
    }
```

Output:

Signature of the Faculty