# **Design and implement a stack (Array implementation/ Linked list implementation) and demonstrate its working with necessary inputs. Display the appropriate messages in case of exceptions.**

**THEORY**

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

**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 −

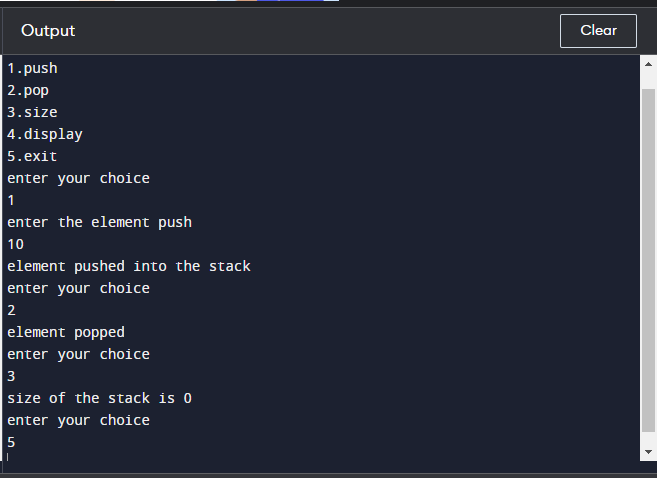
* **peek()** − get the top data element of the stack, without removing it.
* **isFull()** − check if stack is full.
* **isEmpty()** − check if stack is empty.

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 top value of the stack without actually removing it.

**CODE WITH COMMENTS**

#include<stdio.h>//header file  
#include<stdlib.h>  
#define ss 3 //universal declaration of stack size 3  
  
int mstack[3], top=-1;//universal variable declaration  
  
void push(int ele)//function to push the element  
{  
    if(top==ss-1)//condition for stack overflow  
    {  
        printf("stack overflow\n");//prints stack overflow  
        return;  
    }  
    top++;  
    mstack[top]=ele;//element pushed into the stack  
    printf("element pushed into the stack\n");  
}  
  
int pop()//function to pop the element  
{  
    if(top==-1)  
    {  
        printf("stack is empty\n");//prints stack is empty  
        return -1;  
    }  
    int ele=mstack[top];//element popped into the stack  
    top--;  
    printf("element popped\n");//prints element is popped  
}  
  
void size()//function to display the size of the stack  
{  
    printf("size of the stack is %d\n",top+1);//prints stack size  
}  
  
void display()//function to display the stack  
{  
    int i;//variable declaration  
    printf("elements are\n");  
    for(i=0;i<=top-1;i++)//loop to print the stack  
        printf("%d",mstack[i]);//prints the stack element  
}  
  
void main()//main function  
{  
    int choice, ele;//variable declaration  
    printf("1.push\n2.pop\n3.size\n4.display\n5.exit\n");//operations that can be performed  
    while(1)  
    {  
        printf("enter your choice\n");//choice to be entered  
        scanf("%d",&choice);//entered choice is scanned  
        switch(choice)//switch case of choice  
        {  
            case 1: printf("enter the element push\n");//element push  
                scanf("%d",&ele);//scan the element entered  
                push(ele);//push element called  
                break;  
            case 2: pop();//pop function called  
                break;  
            case 3: size();//size function called  
                break;  
            case 4: display();//display function called  
                break;  
            case 5: exit(0);//exits from the loop  
        }  
    }

**OUTPUT**

****

**ALGORITHM**

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 1-D array with fixed size

Step 4:Define an integer variable ‘top’ and initialize it with ‘-1’

Step 5:In main function,display more with list of operations and make a suitable function call to perform operation selected by the user on the stack

To push:

Step 1:check whether stack is FULL

Step 2:If it is full,then display stack is full

Step 3:If it is not full,then increment top value by one and set stack(top) to value (stack(top)=value)

To pop()

Step 1:Check whether the stack is empty ,if its not empty delete stack[top]and decrement top by one

To display:

Step 1:Check whether the stack is empty

Step 2:If empty display”Stack is empty”

Step 3:If not empty,display stack[1]value and decrement i value by one