Date-

Assignment No. :

Problem Statement:

Program in C to implement stack using array.

Theory:

Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack and the only element that can be removed is the element that is at the top of the stack, just like a pile of objects.

Basic features of Stack

1. Stack is an ordered list of similar data type.
2. Stack is a LIFO(Last in First out) structure or we can say FILO(First in Last out).
3. push() function is used to insert new elements into the Stack and pop() function is used to remove an element from the stack. Both insertion and removal are allowed at only one end of Stack called Top.
4. Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.

Applications of Stack

The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.

There are other uses also like:

1. Parsing
2. Expression Conversion(Infix to Postfix, Postfix to Prefix etc)

Algorithm:

Input specification: An array say **a[]**, which will contain the stack,
a variable, say top which will contain the top element of the stack and another variable say s, which will contain the size of the stack.

Output specification: Successful occurrence of the operations of the stack.

Steps:

Algorithm for method ins():

Step 1 If(top==s-1) Then
   a. Print "Stack overflow"
Step 2 Else
   a. Print "Enter the element to insert: "
   b. top=top+1
   c. Input a[top]

Algorithm for method del():

Step 1 If(top<0) Then
   i. Print "Stack underflow"
Step 2 Else
   a. Print "The deleted element is" a[top]
   b. top=top-1

Algorithm for method disp()

Step 1If(top<0)
   a. Print "Stack underflow"
Step 2 Repeat Step 2.a to 2.b For i=top to i>=0
   a. Print a[i]
   b. i=i-1

Algorithm for method main()

Step 1 Print "Enter the size of the stack: "

a. Input s

Step 2 While(TRUE)

a. Print “1.Insertion 2.Deletion 3.Traverse 4.Exit”
b. Print “Enter your choice: ");
c. Input ch
d. If(ch=1)Then
    i. ins()
    ii. break
e. Else If(ch=2)Then
    i. del()
    ii. break
f. Else If(ch=3)Then
    i. disp()
    ii. break
g. Else If(ch=4)Then
    i. Return
h. Else
    i. Print "Wrong choice"
    ii. Break

Source Code:

```
#include<stdio.h>
#include<conio.h>
void ins(void);
void del(void);
void disp(void);
int a[30], s, top=-1;
int main()
{
        int ch;
        printf("Enter the size of the stack: ");
        scanf("%d",&s);

        while(1)
```

```
	{
		printf("\n1.Insertion\n2.Deletion\n3.Traverse\n4.Exit\nEnter your choice: ");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1: ins();
				break;
			case 2: del();
				break;
			case 3: disp();
				break;
			case 4: return 0;
			default: printf("Wrong choice");
		}
	}
	return 0;
}
void ins()
{
		if(top==s-1)
			printf("Stack overflow");
		else{

		printf("Enter the element to insert: ");
		scanf("%d",&a[++top]);
	}
}
void del()
{
	if(top<0)
		printf("Stack underflow");

	else{

		printf("The deleted element is %d",a[top]);
		top--;
```

```
        }
}
void disp()
{
        int i;
        if(top<0)
                printf("Stack underflow");
        for(i=top;i>=0;i--)
                printf("%d\n",a[i]);
}
```

Input & Output:

```
Enter the size of the stack: 4

1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 1
Enter the element to insert: 12

1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 1
Enter the element to insert: 13

1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 1
Enter the element to insert: 15

1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 1
Enter the element to insert: 17

1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 1
Stack overflow
1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 2
The deleted element is 17
The deleted element is 15
1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 3
13
12

1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 2
The deleted element is 13
1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 2
The deleted element is 12
1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 2
Stack underflow
1.Insertion
2.Deletion
3.Traverse
4.Exit
Enter your choice: 3
```

Discussion:

1. Above code is in way a good code. There are lot of problems, to start with pop returns -1 when stack is empty which means that -1 cannot be a valid member of the stack.
2. Push function should ideally return something to tell its calling function whether or not it was able to insert element in the stack. printf is not a good way in production code, because most often it ends up printing in logs. The size of Array is 100, if we just need 10 element stack, then 90 element memory will be wasted. In other case.
3. If we want a stack with 500 element, we will have to change MAX value, which means program need to be compiled again. The solution to this can be to get the size of stack from the calling function and allocate memory using malloc (or new in C++). But then we should be cautious about deallocating memory when done, else it will leave memory leaks.
4. All the above operations are constant time operations and takes O(1) time. In fact, almost all the operations of Stack and Queue are constant time operations, except for operations like printing all elements, etc