

PART A

(PART A : TO BE REFERRED BY STUDENTS)

Experiment No.01

- **Aim:**

Implementation of application of stack.

- **Prerequisite:**

C programming

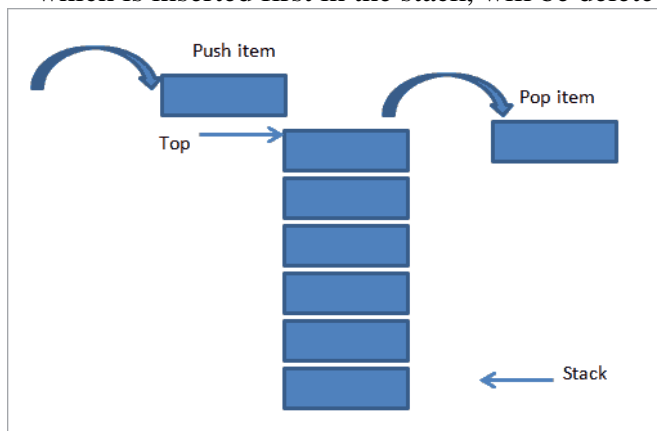
- **Outcome:**

After successful completion of this experiment students will be,
Implement various operations using linear data structures.

- **Theory:**

Stack:

Stack is an ordered, linear list in which, insertion and deletion can be performed only at one end that is called top. Stack is a recursive data structure having pointer to its top element. Stacks are sometimes called as Last-In-First-Out (LIFO) lists i.e. the element which is inserted first in the stack, will be deleted last from the stack.



Basic Stack Operations:

- **Push:** To insert data on the top of the stack
- **Pop:** To remove data from the top of the stack
- **Stack Top:** To get the topmost data from the stack
- **Stack Empty:** To check whether stack is empty
- **Stack full:** To check whether stack is full

Example:

The screenshot shows a Google Slides presentation titled "Module 1 - Stack". The current slide is titled "APPLICATIONS OF STACK" and contains the text "Reversing a list" and "Example: Push 1, 2, 3, 4, 5 and pop all". Below the text is a diagram illustrating the process of reversing a list using a stack. The diagram shows a sequence of operations: Push 1, Push 2, Push 3, Push 4, Push 5, Pop, Pop, Pop, Pop, Pop. The stack grows from bottom to top, and the elements are popped from top to bottom, resulting in the reversed list 5 4 3 2 1.

Algorithm:

Implementation of stack using Array:

Declare array stack with size MAX

1. Push:

Step 1: Read data

Step 2: if $\text{top} = n-1$ then

Print stack is full and go to step 3

else

$\text{top}++$

Set $\text{stack}[\text{top}] = \text{data}$

Step 3: Exit

Pop:

Step 1: if $\text{top} = -1$ then

Print stack is empty and go to step 2

else

Set $\text{data} = \text{stack}[\text{top}]$

$\text{top}--$

return top

Step 2: Exit

Stack Top:

Step 1: if $\text{top} = -1$ then

```

        Print stack is empty and go to step 2
    else
        Set data = stack[top]
        Print data
Step 2: Exit

```

Stack Empty:

```

Step 1: if top = -1 then
    Print stack is empty and go to step 2
else
    Print stack is not empty and go to step 2
Step 2: Exit

```

Stack Full:

```

Step 1: if top = n-1 then
    Print stack is full and go to step 2
else
    Print stack is not full and go to step 2
Step 2: Exit

```

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll. No. A56	Name: ANAS MUBEEN EDROOS
Class: DSE-MTRX	Batch: DSE
Date of Experiment: 13/2/2021	Date of Submission: 20/2/2021
Grade:	

B.1 Software Code written by student:

```

Program:-
//ANAS EDROOS
//Roll No 56

```

```

#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

struct Stack {
    int top;
    unsigned capacity;
    int* array;
};

struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int*)malloc(stack->capacity * sizeof(int));
    return stack;
}

int isFull(struct Stack* stack)
{
    return stack->top == stack->capacity - 1;
}

1
int isEmpty(struct Stack* stack)
{
    return stack->top == -1;
}

void push(struct Stack* stack, int item)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = item;
    printf("%d pushed to stack\n", item);
}

int pop(struct Stack* stack)
{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top--];
}

int peek(struct Stack* stack)
{

```

```

    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top];
}

int main()
{
    struct Stack* stack = createStack(100);

    push(stack, 10);
    push(stack, 20);
    push(stack, 30);

    printf("%d popped from stack\n", pop(stack));

    return 0;
}

```

B2. Output:

```

Enter Choice:3
Elements:
20
30
70
Enter Choice:2
Popped Element: 20
Enter Choice:2
Popped Element: 30
Enter Choice:2
Popped Element: 70
Enter Choice:2
Underflow or Stack is Empty
Enter Choice:3
The STACK is empty
Enter Choice:1
Enter Element to be Pushed:20

Enter Choice:1
Enter Element to be Pushed:40

Enter Choice:1
Enter Element to be Pushed:5

Enter Choice:3
Elements:
5
40
20
Enter Choice:

```

B3. Observations and learning:

We observe how implementation of stacks are done.

B4. Conclusion:

We have understood the implementation of Stack.

B5. Question of Curiosity

1) Explain stack operations using Linked List.

soln. push() : Insert the element into linked list nothing but which is the top node of Stack.

pop() : Return top element from the Stack and move the top pointer to the second
node of linked list or Stack.

peek(): Return the top element.

display(): Print all element of Stack.

2) Compare stack and queue.

Soln.

Stacks	Queues
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
Insert operation is called push operation.	Insert operation is called enqueue operation.
Delete operation is called pop operation.	Delete operation is called dequeue operation.

3) Write applications of stack.
soln.

- Stacks can be used for expression evaluation.
- Stacks can be used to check parenthesis matching in an expression.
- Stacks can be used for Conversion from one form of expression to another.
- Stacks can be used for Memory Management.
- Stack data structures are used in backtracking problems.