PART A

(PART A : TO BE REFERED BY STUDENTS)

Experiment No.01

A.1 Aim:

Implementation of application of stack.

A.2 Prerequisite:

C programming

A.3 Outcome:

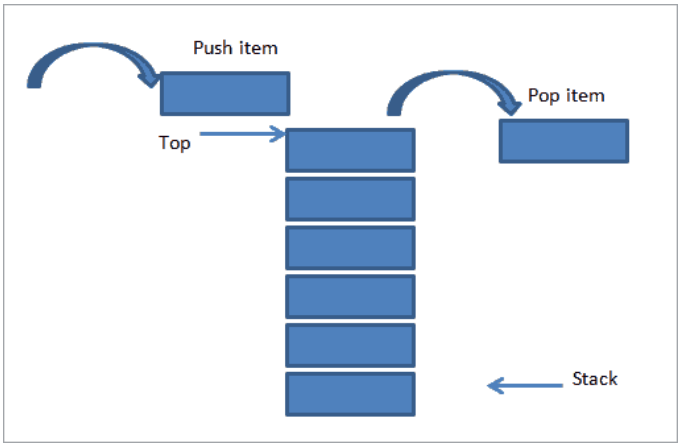
After successful completion of this experiment students will be,

Implement various operations using linear data structures.

A.4 Theory:

Stack:

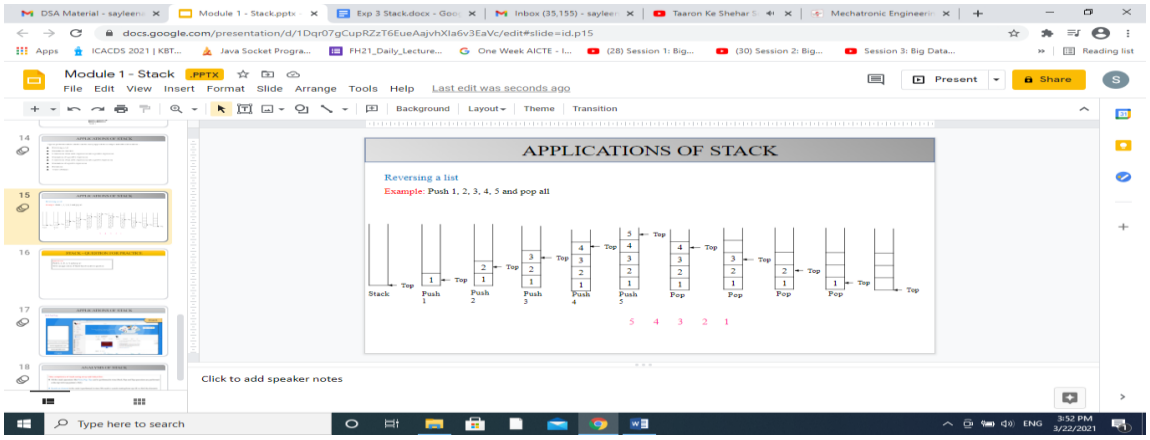
Stack is an ordered, linear list in which, insertion and deletion can be performed only at one end that is called top. Stack is a recursive data structure having pointer to its top element. Stacks are sometimes called as Last-In-First-Out (LIFO) lists i.e. the element which is inserted first in the stack,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



Algorithm:
Implementation of stack using Array:

Declare array stack with size MAX

1. **Push:**

Step 1: Read data
Step 2: if top = n-1 then
Print stack is full and go to step 3
else
top++
Set stack[top] = data
Step 3: Exit

326115432. **Pop:**

Step 1: if top = -1 then
Print stack is empty and go to step 2
else
Set data = stack[top]
top--
return top
Step 2: Exit

326115600. **Stack Top:**

Step 1: if top = -1 then
Print stack is empty and go to step 2
else

Set data = stack[top]
Print data
Step 2: Exit

326115488. **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

326115208. **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.71	Name: Omkar Gujja
Class: 2	Batch: MTRX
Date of Experiment: 13/2/2021	Date of Submission: 20/2/2021
Grade:	

B.1 Software Code written by student:

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

#define Size 4

int Top=-1, inp_array[Size];
```

```
void Push();
void Pop();
void isEmpty();
void isFull();
void show();

int main()
{
   int choice;
   while(1)
   {
      printf("\nOperations performed by Stack");
      printf("\n1.Push the element\n2.Pop the element\n3.Is
Empty?\n4.Is Full\n5.Show\n6.End");
      printf("\n\nEnter the choice:");
      scanf("%d",&choice);

      switch(choice)
      {
         case 1: Push();
               break;
         case 2: Pop();
               break;
         case 3: isEmpty();
               break;
         case 4: isFull();
               break;
         case 5: show();
               break;
         case 6: exit(0);

         default: printf("\nInvalid choice!!");
      }
   }
}
```

```
void Push()
{
   int x;

   if(Top==Size-1)
   {
      printf("\nOverflow!!");
   }
   else
   {
      printf("\nEnter element to be inserted to the stack:");
      scanf("%d",&x);
      Top=Top+1;
      inp_array[Top]=x;
   }
}

void Pop()
{
   if(Top==-1)
   {
      printf("\nUnderflow!!");
   }
   else
   {
      printf("\nPopped element:  %d",inp_array[Top]);
      Top=Top-1;
   }
}

void isEmpty(){
   if (Top == -1){
      printf("\nStack is Empty!!");
   }
```

```
   else{
      printf("\nStack is not Empty!!");
   }
}
void isFull(){
   if (Top == Size-1){
      printf("\nStack is Full!!");
   }
   else{
      printf("\nStack is not Full!!");
   }
}

void show()
{


   if(Top==-1)
   {
      printf("\nUnderflow!!");
   }
   else
   {
      printf("\nElements present in the stack: \n");
      for(int i=Top;i>=0;--i)
         printf("%d\n",inp_array[i]);
   }
}
```

B.2 Input and Output:

```
Operations performed by Stack
1.Push the element
2.Pop the element
3.Is Empty?
4.Is Full
5.Show
6.End

Enter the choice:1

Enter element to be inserted to the stack:2

Operations performed by Stack
1.Push the element
2.Pop the element
3.Is Empty?
4.Is Full
5.Show
6.End

Enter the choice:4

Stack is not Full!!
```

```
Operations performed by Stack
1.Push the element
2.Pop the element
3.Is Empty?
4.Is Full
5.Show
6.End

Enter the choice:5

Elements present in the stack:
2

Operations performed by Stack
1.Push the element
2.Pop the element
3.Is Empty?
4.Is Full
5.Show
6.End

Enter the choice:6


...Program finished with exit code 0
Press ENTER to exit console.
```

B.3 Observations and learning:

We observe how implemation of stacks are done

B.4 Conclusion:

We have understood the implementation of Stack

B.5 Question of Curiosity

Q.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

Q.2 Compare stack and queue.

#	STACK	QUEUE
1	Objects are inserted and removed at the same end.	Objects are inserted and removed from different ends.
2	In stacks only one pointer is used. It points to the top of the stack.	In queues, two different pointers are used for front and rear ends.
3	In stacks, the last inserted object is first to come out.	In queues, the object inserted first is first deleted.
4	Stacks follow Last In First Out (LIFO) order.	Queues following First In First Out (FIFO) order.
5	Stack operations are called push and pop.	Queue operations are called enqueue and dequeue.
6	Stacks are visualized as vertical collections.	Queues are visualized as horizontal collections.
7	Collection of dinner plates at a wedding reception is an example of stack.	People standing in a file to board a bus is an example of queue.

Q.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.