Experiment No.1
Aim - Implement Stack ADT using array.
Name: Shravani Sandeep Raut
Roll no: 48
Date of Performance:
Date of Submission:

Experiment No. 1: To implement stack ADT using arrays

Aim: To implement stack ADT using arrays.

Objective:

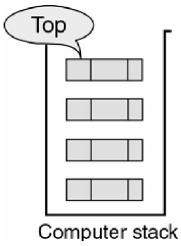
1) Understand the Stack Data Structure and its basic operators.

2) Understand the method of defining stack ADT and implement the basic operators.

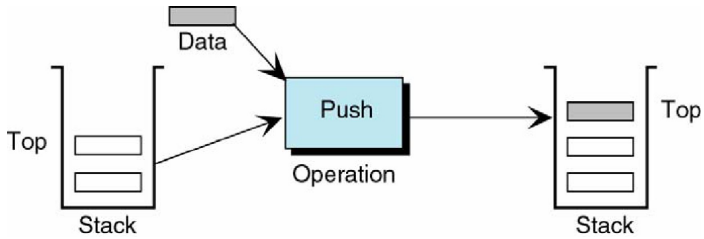
3) Learn how to create objects from an ADT and invoke member functions.

Theory:

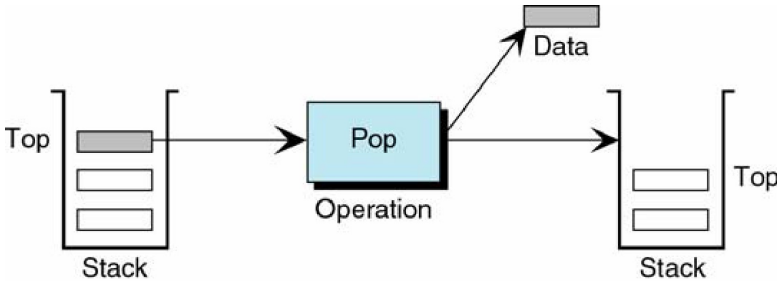
A stack is a data structure where all insertions and deletions occur at one end, known as the top. It follows the Last In First Out (LIFO) principle, meaning the last element added to the stack will be the first to be removed. Key operations for a stack are "push" to add an element to the top, and "pop" to remove the top element. Auxiliary operations include "peek" to view the top element without removing it, "isEmpty" to check if the stack is empty, and "isFull" to determine if the stack is at its maximum capacity. Errors can occur when pushing to a full stack or popping from an empty stack, so "isEmpty" and "isFull" functions are used to check these conditions. The "top" variable is typically initialized to -1 before any insertions into the stack.



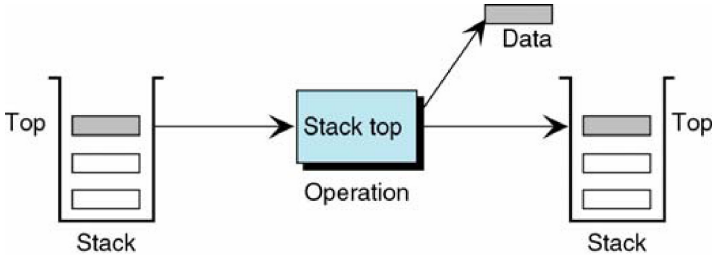
Push Operation



Pop Operation



Peek Operation



Algorithm:

PUSH(item)

1. If (stack is full)

   Print "overflow"

2. top = top + 1
3. stack[top] = item

   Return

POP()

1. If (stack is empty)

   Print "underflow"

2. Item = stack[top]
3. top = top – 1
4. Return item

PEEK()

1. If (stack is empty)

Print "underflow"

2. Item = stack[top]

3. Return item

ISEMPTY()

1. If(top = -1)then

return 1

2. return 0

ISFULL()

1. If(top = max)then

return 1

2. return 0

Code:

```
//Array implementation of Stack
#include <stdio.h>
#include <conio.h>
#define MAX 100

int a[MAX], n, i, top= -1;

void push()
{
   if(top == MAX -1)
   {
      printf("Stack is full");
   }
   else
   {
      printf("Enter the element to be inserted: ");
      scanf("%d", &n);
      top++;
      a[top] = n;
   }
}

void pop()
{
   if(top == -1)
   {
```

```
        printf("Empty stack");
    }
    else
    {
        a[top] = n;
        printf("Enter no deleted is %d ",n);
        top--;
    }
}

void display()
{
    if(top == -1)
    {
        printf("Empty stack");
    }
    else
    {
        for(i=0; i<= top; i++)
        {
            printf("%d\t",a[i]);
        }
    }
}

void peek()
{
    a[top] = n;
    printf("The top element is %d", n);
}


void main()
{
    int m=0;
    do{
    printf("\nOperations on stack");
    printf("\n1. PUSH");
    printf("\n2. POP");
    printf("\n3. DISPLAY");
    printf("\n4. PEEK");
    printf("\n5. EXIT");
    printf("\nEnter your choice: ");
    scanf("%d", &m);
```

```
  switch(m)
  {
        case 1: push();
        break;
        case 2: pop();
        break;
        case 3: display();
        break;
        case 4: peek();
        break;
  }
  }while(m != 5);
}
```

Output:

```
Operations on stack
1. PUSH
2. POP
3. DISPLAY
4. PEEK
5. EXIT
Enter your choice: 1
Enter the element to be inserted: 10

Operations on stack
1. PUSH
2. POP
3. DISPLAY
4. PEEK
5. EXIT
Enter your choice: 2
Enter no deleted is 10
```

Conclusion:

1) What is the structure of Stack ADT?

Stack ADT consists of

- Push(): Adds element to the top of the stack.
- Pop(): Removes and returns the top element of the stack.
- Peek(): Returns the top element of the stack without removing it.
- isEmpty(): Returns true if the stack is empty, otherwise false.
- IsFull(): Returns true if stack is full
- Display(): Returns the elements in stack

2) List various applications of stack?

1. Wellformness of parenthesis
2. Conversion – Infix to Postfix
3. Evalutaion of Postfix expression
4. Reversing
5. Recursion

3) Which stack operation will be used when the recursive function call is returning to the calling function?

- When a recursive function call is returning to the calling function, the stack operation used is "pop."
- **Push**: When a function is called (including a recursive call), the current execution context (including local variables, return address, etc.) is saved onto the call stack. This operation is called "push."
- **Pop**: When the function completes its execution and is about to return to the calling function, the top of the call stack is removed. This operation is called "pop." The return value, if any, is also sent back to the caller during this process.