Experiment No.1
Implement Stack ADT using array.
Name: Singh Rahul Rammilan
Roll no: 56
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 1: To implement stack ADT using arrays

Aim: To implement stack ADT using arrays.

Objective:

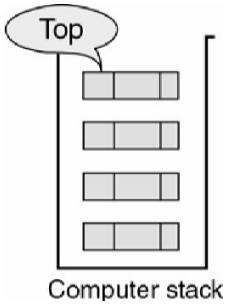
- Understand the Stack Data Structure and its basic operators.
- Understand the method of defining stack ADT and implement the basic operators.
- Learn how to create objects from an ADT and invoke member functions.

Theory:

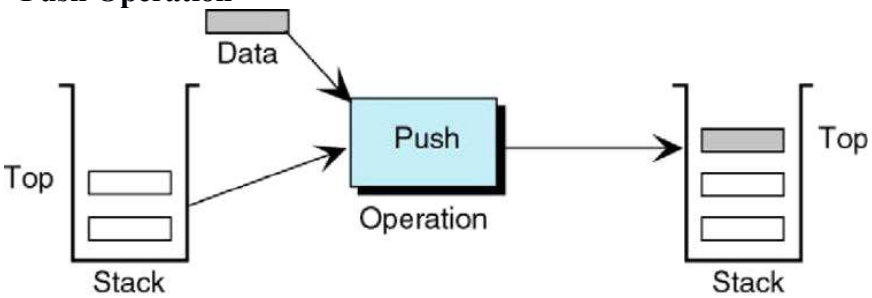
A stack is a data structure where all insertions and deletions occur at one end, known as the top. It follows the Last In First Out (LIFO) principle, meaning the last element added to the stack will be the first to be removed. Key operations for a stack are "push" to add an element to the top, and "pop" 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

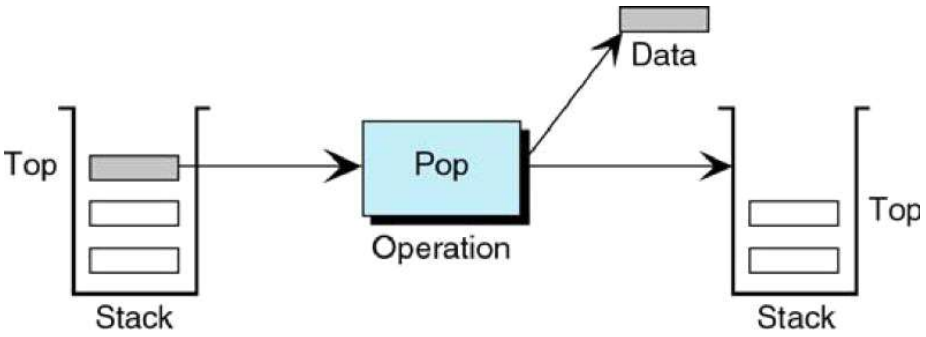
stack, so "isEmpty" and "isFull" functions are used to check these conditions. The "top" variable is typically initialized to -1 before any insertions into the stack.



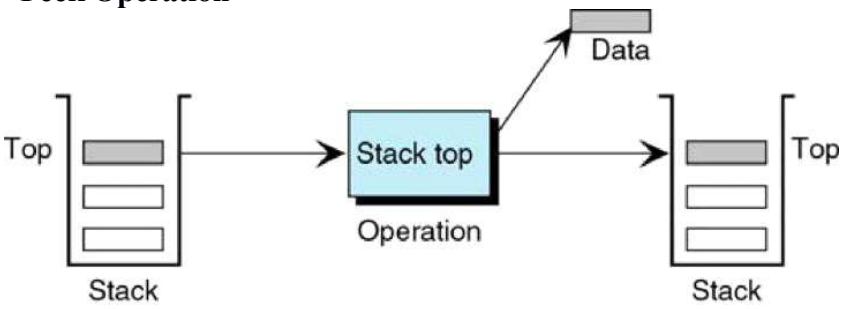
Push Operation



Pop Operation



Peek Operation



Algorithm:

PUSH(item)

- If (stack is full) Print "overflow"
- top = top + 1
- stack[top] = item Return

POP()

- If (stack is empty) Print "underflow"
- Item = stack[top]

- top = top – 1

- Return

item

PEEK()

- If (stack is

  empty) Print

  “underflow”
- Item = stack[top]

- Return

item

ISEMPTY(

)
- If(top = -1)then

  return 1

- retu

rn 0

ISFUL

L()
- If(top = max)then

  return 1

- return 0

Code:

```
#include<stdio.h>

#include<stdlib.h>
```

```
int
stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void
display(void);
void peek();

int main()
{

top=-1;
clrscr();
printf("Enter the size of stack[max=100]:"); scanf("%
d",&n);
printf("Stack operation using array\n");

printf("\n\t 1.PUSH \n\t 2.POP \n\t 3.PEEK \n\t 4.DISPLAY \n\t
        5.EXIT"); do
        {

                printf("\nEnter your choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1:
                        {
```

```
            push();
            break;
}
case 2:
{
            pop
            ();
            bre
            ak;
}
case 3:
{
            peek();
            break;
}
case 4:
{
            display();
            break;
}
case 5:
{
            printf("\n\tEXIT
            POINT"); break;
}
default:
```

```
                {
                        printf("\n\t Please enter a valid choice(1/2/3/4)");
                }
        }
    }
    while(choice!=5);
return 0;
}

void push()
{
    if(top>=n-1)
    {
        printf("\n\t Stack is 'OVERFLOW' ");
    }
    else
    {
        printf("\n Enter a value to be
        pushed:"); scanf("%d",&x);
        top++;
        stack[top]=x
        ;
    }
}
void pop()
```

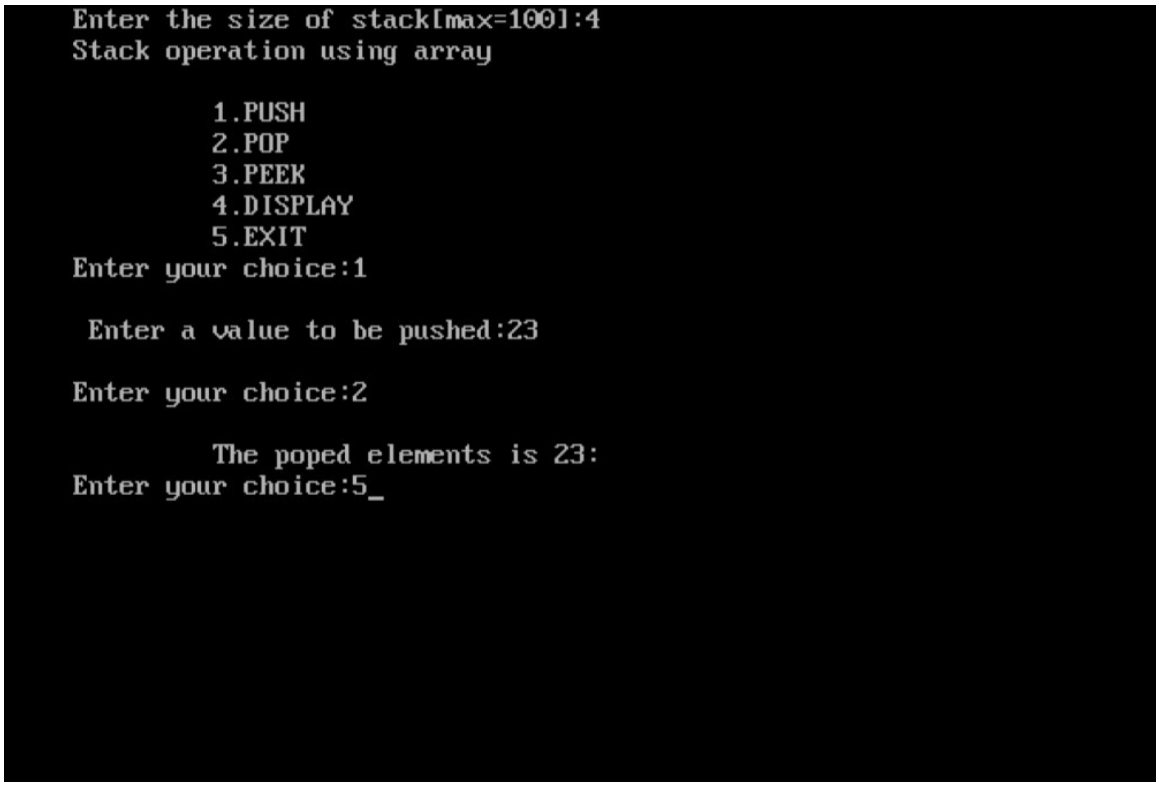
```
{
        if(top<=-1)
        {
                printf("\nStack is 'UNDERFLOW' ");
        }
        else
        {
                printf("\n\t The poped elements is %
                d:",stack[top]); top--;
        }
}
void display()
{
        if(top>=0)
        {
                printf("\n The element in
                stack:"); for(i=top;i>=0;i--)
                {
                        printf("\n%d",stack[i]);
                        printf("\nPress next choice");
                }
        }
        else
        {
                printf("\nThe stack is empty");
```

```
        }


}
void peek()
{
        if(top<=-1)
        {
                printf("\n stack is Underflow");
        }
        else
        {
                printf("\n The peek element is %d:",stack[top]);
        }


}
```

Output:



Conclusion:

- What is the structure of Stack ADT?
- The structure of a Stack Abstract Data Type (ADT) is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It consists of a collection of elements with two main operations: push (to add an element to the top of the

  stack) and pop (to remove the top element from the stack). Additionally, stacks typically have operations like peek (to view the top element without removing it) and isEmpty (to check if the stack is empty).

- List various applications of stack?

- Various applications of stacks include:
  - Function call management in programming (used for function call stack during recursion).
  - Undo mechanisms in software applications.
  - Expression evaluation (e.g., evaluating arithmetic expressions with parentheses).
  - Memory management in computer systems (used for call stacks).
  - Parsing and syntax analysis in compilers.
  - Backtracking algorithms (e.g., solving puzzles like Sudoku or mazes).
  - Handling of browser history in web browsers.

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

- The stack operation used when a recursive function call is returning to the calling function is "pop." When a function returns, the top of the function call stack is popped, allowing the program to resume execution in the calling function. This is how the recursion stack works to manage the order of function calls and their return values in a LIFO fashion.