Experiment No.1
Implement Stack ADT using array.
Name: Gautam D. Chaudhari
Roll no: 04
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:

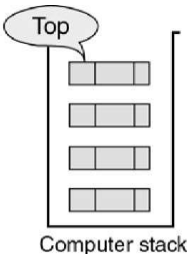
1) Understand the Stack Data Structure and its basic operators.

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

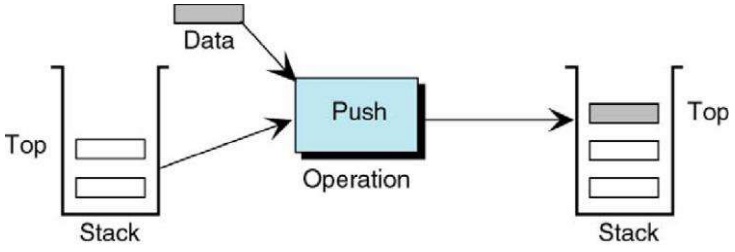
3) Learn how to create objects from an ADT and invoke member functions.

Theory:

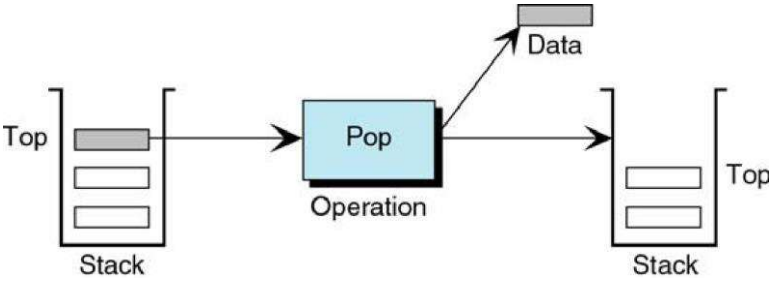
A stack is a data structure where all insertions and deletions occur at one end, known as the top. It follows the Last In First Out (LIFO) principle, meaning the last element added to the stack will be the first to be removed. Key operations for a stack are "push" to add an element to the top, and "pop"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



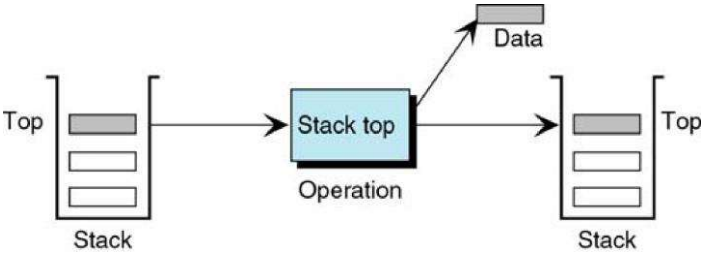
Push Operation



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:

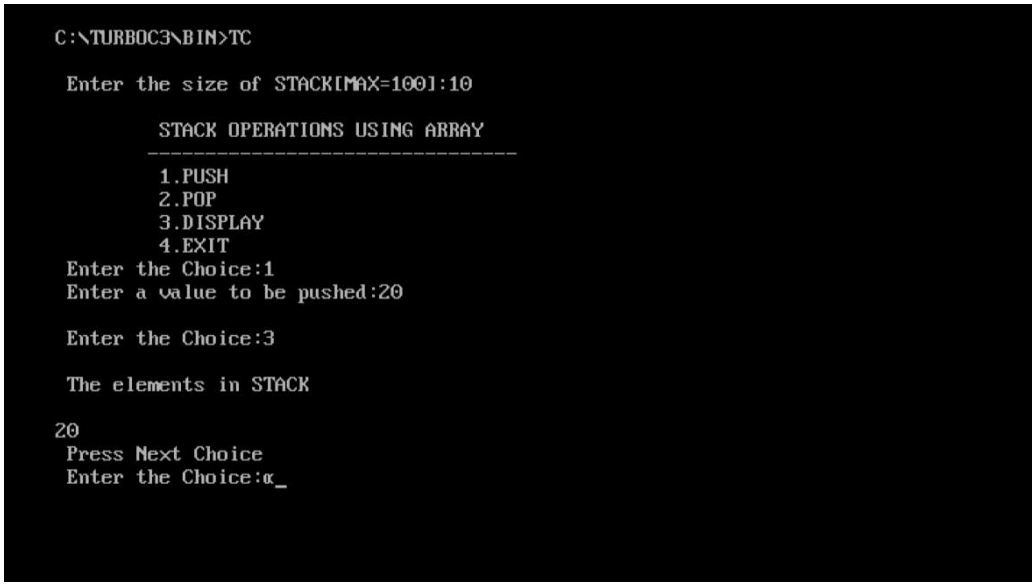
```
#include<stdio.h>

int stack[100],choice,n,top,x,i;


void push(void);

void pop(void);

void display(void);


int main() {

   top = -1;

   printf("\n Enter the size of STACK[MAX=100]:");

   scanf("%d", &n);

   printf("\n\t STACK OPERATIONS USING ARRAY");

   printf("\n\t--------------------------------");

   printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
```

```
do {
   printf("\n Enter the Choice:");
   scanf("%d", &choice);

   switch (choice) {
      case 1: {
         push();
         break;
      }
      case 2: {
         pop();
         break;
      }
      case 3: {
         display();
         break;
      }
      case 4: {
         printf("\n\t EXIT POINT ");
         break;
      }
      default: {
         printf("\n\t Please Enter a Valid Choice(1/2/3/4)");
      }
   }
```

```
    } while (choice != 4);


    return 0;
}


void push() {
    if (top >= n - 1) {
        printf("\n\tSTACK is over flow");
    } else {
        printf(" Enter a value to be pushed:");
        scanf("%d", &x);
        top++;
        stack[top] = x;
    }
}


void pop() {
    if (top <= -1) {
        printf("\n\t Stack is under flow");
    } else {
        printf("\n\t The popped elements is %d", stack[top]);
        top--;
    }
}


void display() {
```

```
  if (top >= 0) {
     printf("\n The elements in STACK \n");
     for (i = top; i >= 0; i--)
        printf("\n%d", stack[i]);
     printf("\n Press Next Choice");
  } else {
     printf("\n The STACK is empty");
  }
}
```

Output:



Conclusion:

What is the structure of Stack ADT?

A stack is a linear data structure primarily defined by its key operations. It allows you to "Push" elements onto the top of the stack and "Pop" elements, which removes and returns the top element. You can also "Peek" at the top element without removing it, check if the stack is empty with "isEmpty," and determine the number of elements using "Size." The stack adheres to the Last-In-First-Out (LIFO) principle, which governs the order in which elements are accessed and removed.

List various applications of stack?

- **Function Call Management:** Stacks keep track of function calls, allowing for nested function execution and easy return to previous function states.
- **Expression Evaluation:** Stacks help in parsing and evaluating mathematical expressions, converting infix expressions to postfix or prefix notation, making calculations, and maintaining operator precedence.
- **Memory Management:** Stacks are instrumental in managing program memory, especially in the allocation and deallocation of memory for local variables and function parameters.
- **Backtracking:** Stacks aid in backtracking algorithms, where you can store and retrieve previous states or choices.
- **Undo Mechanisms:** Many applications use stacks to implement undo and redo functionality by saving previous states.
- **Browser History:** Web browsers often use stacks to keep track of the pages you've visited, allowing you to go back or forward in your browsing history.

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 returns to the calling function is "Pop."
In a recursive function, each function call creates a new stack frame (or activation record) to store its local variables and return address. When the recursive function completes its task and needs to return to the calling function, the "Pop" operation is performed to remove the current stack frame and restore the program's state to the state of the calling function. This process continues until the initial calling function (the one that started the recursion) is reached, at which point the stack becomes empty, and the recursion terminates.