Experiment No. 1: To implement stack ADT using arrays

Aim: To implement stack ADT using arrays.

Objective:

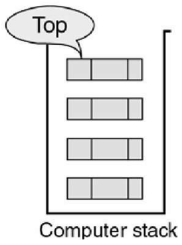
1) Understand the Stack Data Structure and its basic operators.

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

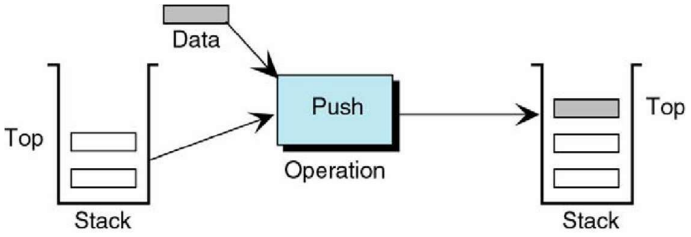
3) Learn how to create objects from an ADT and invoke member functions.

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.

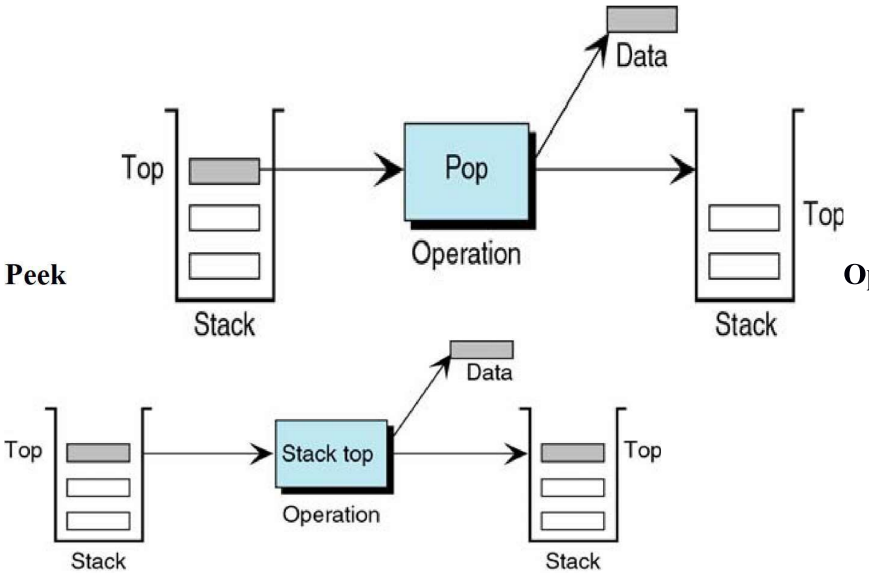


Push Operation



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

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 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);}
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: Enter the size of STACK[MAX=100]:5

STACK OPERATIONS USING ARRAY

1.PUSH

2.POP

3.DISPLAY

4.EXIT

Enter the Choice:1

Enter a value to be pushed:10

Enter the Choice:1

Enter a value to be pushed:20

Enter the Choice:1

Enter a value to be pushed:0

Enter the Choice:2

The popped elements is 0

Enter the Choice:3

The elements in STACK :20 10

Enter the Choice:4

Conclusion:

What is the structure of Stack ADT?

- The structure of stack ADT is a way of organizing data in a linear fashion, such that the operations of insertion and deletion are performed at one end, called the top.
- The stack ADT follows the principle of LIFO (Last In First Out), which means that the last element inserted is the first one to be removed.
- he stack ADT supports the following operations: Push , Pop , Peek , isEmpty , isFull.
- The stack ADT can be implemented using different data structures, such as arrays or linked list
- Depending on the implementation, the stack ADT may have a fixed or variable size. The stack ADT can be used for various applications, such as reversing a string, evaluating expressions, converting numbers between bases, etc.

List various applications of stack?

- Evaluating expressions: A stack can be used to evaluate expressions that consist of operands and operators, such as infix, postfix, and prefix notations. The stack can store the operands and perform the operations in the correct order
- Backtracking: A stack can be used to implement backtracking algorithms, such as finding a path in a maze, solving a sudoku puzzle, or generating permutations. The stack can store the current state of the problem and backtrack to a previous state if needed
- Undo/redo operations: A stack can be used to implement the undo/redo feature in various applications, such as text editors, browsers, or games. The stack can store the previous actions performed by the user and undo or redo them as required.

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

The stack operation that will be used when the recursive function call is returning to the calling function is **pop**. This means that the topmost stack frame, which contains the data of the current recursive call, is removed from the stack and the control is returned to the previous function call. The pop operation also updates the program counter, which holds the address of the next instruction to be executed, to the return address stored in the stack frame. This way, the recursive function can resume its execution from where it left off before making another recursive call.