Experiment No.1
Implement Stack ADT using array.
Name: Sharvari Anand Bhondekar
Roll no: 06
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:

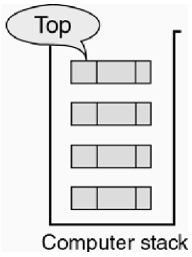
1) Understand the Stack Data Structure and its basic operators.

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

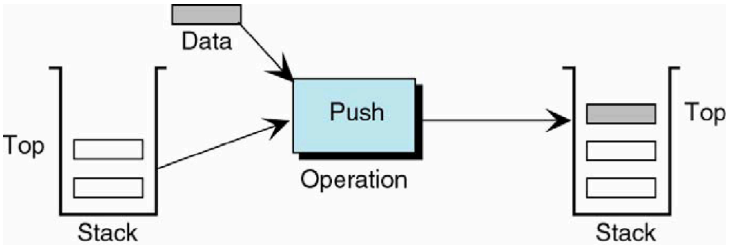
3) Learn how to create objects from an ADT and invoke member functions.

Theory:

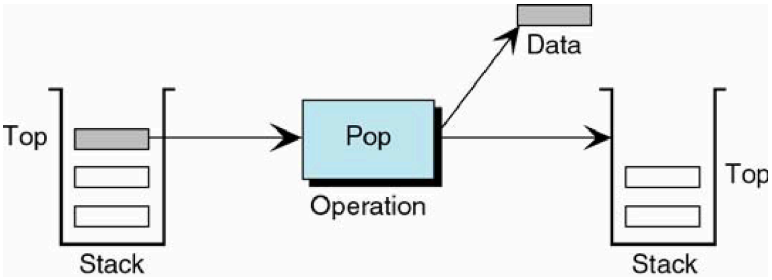
A stack is a data structure where all insertions and deletions occur at one end, known as the top. It follows the Last In First Out (LIFO) principle, meaning the last element added to the stack will be the first to be removed. Key operations for a stack are "push" to add an element to the top, and "pop"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



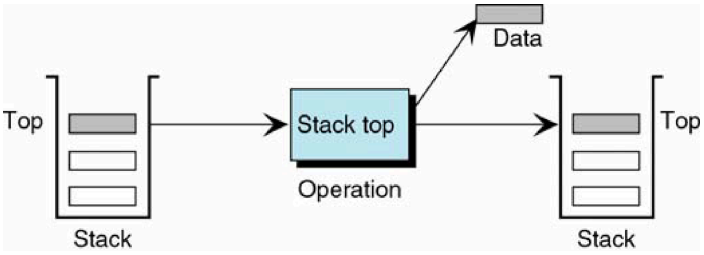
Push Operation



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>
#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;
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();
break;
}
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:

```
Enter the size of stack[max=100]:5
Stack operation using array

          1.PUSH
          2.POP
          3.PEEK
          4.DISPLAY
          5.EXIT
Enter your choice:1

 Enter a value to be pushed:50

Enter your choice:2

          The poped elements is 50:
Enter your choice:3

 stack is Underflow
Enter your choice:1

 Enter a value to be pushed:15

Enter your choice:3

 The peek element is 15:
Enter your choice:4

 The element in stack:
15
Press next choice
Enter your choice:5

         EXIT POINT
```

Conclusion:

1.)What is the structure of Stack ADT?

The Stack Abstract Data Type (ADT) typically consists of a collection of elements with two primary operations:

1. **Push**: Adds an element to the top of the stack.
2. **Pop**: Removes and returns the element from the top of the stack.

Additionally, stacks often include auxiliary operations:

- **Peek (or Top)**: Returns the element at the top without removing it.
- **IsEmpty**: Checks if the stack is empty.
- **Size**: Returns the number of elements in the stack.

The stack operates on a Last In, First Out (LIFO) principle, meaning the most recently added element is the first to be removed. The stack can be implemented using various underlying structures, such as arrays or linked lists.

2.)List various applications of stack?

Here are various applications of stacks:

1. **Function Call Management**: Tracks function calls (call stack).
2. **Expression Evaluation**: Evaluates infix, postfix, and prefix expressions.
3. **Syntax Parsing**: Parses expressions and matches parentheses.
4. **Backtracking**: Supports algorithms like DFS and puzzle solving.
5. **Memory Management**: Manages memory allocation and deallocation.
6. **Undo Mechanisms**: Implements undo features in applications.
7. **Browser History**: Manages navigation history.
8. **Data Reversal**: Reverses strings or data structures.

These utilize the LIFO property of stacks for efficient processing.

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 Pop operation is used. This removes the top function call from the call stack, allowing control to return to the previous function call that was waiting for the result.