**EXPERIMENT NO:1** -**To Implement stack ADT using arrays**

**NAME: TRUSHANT RATHOD**

**ROLL NO: 18**

**BATCH: A DIV: COMPS 3**

**AIM:** To Implement stack ADT using arrays

**OBJECTIVE:**

1) Understand the stack Data structure and its basic operations .

2) understand the method of deficiency stack ADT and implement the basic operation.

3) Learn how to create object from an ADT and invoke member function.

**THEORY:**

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle, meaning the last element added to the stack is the first one to be removed. Stacks are commonly used in computer science and programming for various applications, such as function calls, expression evaluation, parsing, and managing memory.

* Operations:
* Push: Adds an element to the top of the stack.
* 
* Pop: Removes the top element from the stack.
* 
* Peek (or Top): Retrieves the top element without removing it.
* 
* isEmpty: Checks if the stack is empty.
* 
* Representation:
* Stacks can be implemented using arrays or linked lists.
* The top of the stack is the last element of the array or the head of the linked list.
* Stack Overflow:
* If a push operation is attempted on a full stack (in the case of arrays) or if there is no more memory to allocate a new node (in the case of linked lists), it results in a stack overflow error.
* Stack Underflow:
* If a pop operation is attempted on an empty stack, it results in a stack underflow error.

**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. top-1

4. Return item

**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 OPPERATION 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 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\t stack 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:**

In conclusion, the ADT stack is a powerful data structure that offers specific advantages, such as its simplicity, efficient push and pop operations, and its use in managing function calls and parentheses Understanding when and how to use stacks effectively can significantly enhance programming and problem-solving capabilities. Hence we've successfully run the stack ADT using array program.