Experiment – 1

# IMPLEMENTATION OF STACK USING ARRAY

**What is Stack?**

A stack is a fundamental data structure in computer science that follows the Last-In-First-Out (LIFO) principle. It is analogous to a stack of items in the physical world, where the last item added is the first one to be removed. Stacks find extensive application in programming due to their simplicity and efficiency in managing data in a specific order. Here's a concise overview of the stack data structure:

**Basic characteristics of Stack:**

* A stack is a linear data structure that can be thought of as a collection of elements arranged in a linear order.
* Elements can only be added or removed from one end, known as the "top" of the stack.
* The top represents the most recently added item, and the base is the first item added.

**Operations performed on stack:**

* Push: This operation adds an element to the top of the stack.
* Pop: This operation removes the top element from the stack.
* Peek (or Top): This operation returns the value of the top element without removing it.
* isEmpty: This operation checks if the stack is empty.
* isFull: In the context of a fixed-size stack, this operation checks if the stack is full.

**Uses of Stack:**

* Stacks are extensively used in handling function calls and recursive algorithms. As each function call is added to the stack, it's popped off once the function returns.
* They're essential in expression evaluation, like in converting infix expressions to postfix expressions.
* Undo functionality in software applications often uses a stack to maintain a history of operations.
* Stacks are useful for managing system memory, such as storing return addresses in a call stack.

**Implementation of Stack:**

* Stacks can be implemented using arrays or linked lists.
* In an array-based implementation, a fixed-size array holds the stack elements. The top index is adjusted accordingly.
* A linked list-based implementation allows dynamic resizing of the stack. Each node of the linked list holds a value and a reference to the next node.

**Limitations of Stack:**

* Array-based stacks have a fixed size, limiting the number of elements that can be stored. Linked list-based stacks can grow dynamically but have memory overhead.
* Stacks can only be accessed from the top, which can limit certain types of operations.

**Time Complexity of Stack:**

- Push and pop operations in a stack have a time complexity of O(1), which means they have constant time complexity.

- However, resizing an array-based stack or traversing a linked list-based stack would take O(n) time, where n is the number of elements in the stack.

**//program to implement stack using array and include some of its main operations such as push(), pop(), top(), isEmpty(), size()**

CODE:

#include<stdio.h>

#include<stdbool.h>

int top=-1;

int stack[50];

int max=50;

//to check whether the stack is full or not//

bool isFull(){

    if(top==max-1){

        return true;}

    else{

        return false;}

             }

//to check whether the function is empty or not

bool isEmpty()

    {

    if(top==-1){

        return true;

    }

    else{

        return false;

        }

    }

//to push an element into the stack

void push(int data){

    if(!isFull()){

        top++;

        stack[top]=data;

                }

    else{

        printf("Cannot insert data,the stack is full\n");

    }

                 }

//to remove an element from the stack

void pop()

    {

    if(top<=-1)

    {

        printf("\n\t Stack is under flow");

    }

    else

    {

        printf("\n THE POPPED ELEMENTS IS: %d\n",stack[top]);

        top--;

    }

    }

//to see the top element

int peek(){

    return (stack[top]);

}

//to display the entire stack

void display(){

    int i;

    if(isEmpty())

    {

        printf("The stack is empty,cant diplay anything ");

    }

    else{

        printf("The elements of stack are:");

    for(i=top;i>=0;i--){

        printf("%d\t",stack[i]);

    }

    }

    }

//main function

void main(){

    int optn;

    while(optn!=5){

        printf("\nPLEASE ENTER THE OPERATION YOU WANT TO PERFORM\n1.PUSH\n2.POP\n3.PEAK\n4.DISPLAY\n5.EXIT\nChoice:");

        scanf("%d",&optn);

    switch(optn){

        case 1:printf("PLEASE ENTER THE ELEMENT YOU WANT TO PUSH:");

            int num;

            scanf("%d",&num);

            push(num);

            break;

        case 2:

            pop();

            break;

        case 3:

            printf("%d", peek());

            break;

        case 4:

            display();

            break;

    }

    }

    }

/\*OUTPUT:

PLEASE ENTER THE OPERATION YOU WANT TO PERFORM

1.PUSH

2.POP

3.PEAK

4.DISPLAY

5.EXIT

Choice:1

PLEASE ENTER THE ELEMENT YOU WANT TO PUSH:50

PLEASE ENTER THE OPERATION YOU WANT TO PERFORM

1.PUSH

2.POP

3.PEAK

4.DISPLAY

5.EXIT

Choice:1

PLEASE ENTER THE ELEMENT YOU WANT TO PUSH:75

PLEASE ENTER THE OPERATION YOU WANT TO PERFORM

1.PUSH

2.POP

3.PEAK

4.DISPLAY

5.EXIT

Choice:1

PLEASE ENTER THE ELEMENT YOU WANT TO PUSH:40

PLEASE ENTER THE OPERATION YOU WANT TO PERFORM

1.PUSH

2.POP

3.PEAK

4.DISPLAY

5.EXIT

Choice:4

The elements of stack are:50 75 40

PLEASE ENTER THE OPERATION YOU WANT TO PERFORM

1.PUSH

2.POP

3.PEAK

4.DISPLAY

5.EXIT

Choice:2

THE POPPED ELEMENTS IS: 40

PLEASE ENTER THE OPERATION YOU WANT TO PERFORM

1.PUSH

2.POP

3.PEAK

4.DISPLAY

5.EXIT

Choice:4

The elements of stack are:50 75

PLEASE ENTER THE OPERATION YOU WANT TO PERFORM

1.PUSH

2.POP

3.PEAK

4.DISPLAY

5.EXIT

Choice:5

PS C:\Users\OneDrive\Desktop\DSA>

\*/