K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: E2	**Roll No.: 16010123325**
Experiment / assignment / tutorial No. 4	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of Basic operations on stack using Array - Create, Insert, Delete, Peek.

Objective: To implement Basic Operations on Stack i.e. Create, Push, Pop, Peek

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. *https://www.cprogramming.com/tutorial/computersciencetheory/stack.html*
5. *https://www.geeksforgeeks.org/stack-data-structure-introduction-program/*
6. *https://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html*

Abstract:

A Stack is an ordered collection of elements , but it has a special feature that deletion and insertion of elements can be done only from one end, called the top of the stack(TOP). The order may be LIFO(Last In First Out) or FILO(First In Last Out).
Students need to first try and understand the implementation of using arrays. Once comfortable with the concept, they can further implement stacks using linked list as well.

Related Theory: -

Stack is a linear data structure which follows a particular order in which the operations are performed. It works on the mechanism of Last in First out (LIFO).

List 5 Real Life Examples where we use stack:

1. Browser Back and Forward Navigation: Web browsers use stacks to manage the history of web pages you visit. When you navigate to a new page, the current page is pushed onto the back stack. When you press the back button, the page is popped from the stack and displayed. Similarly, a forward stack is used when you navigate forward.

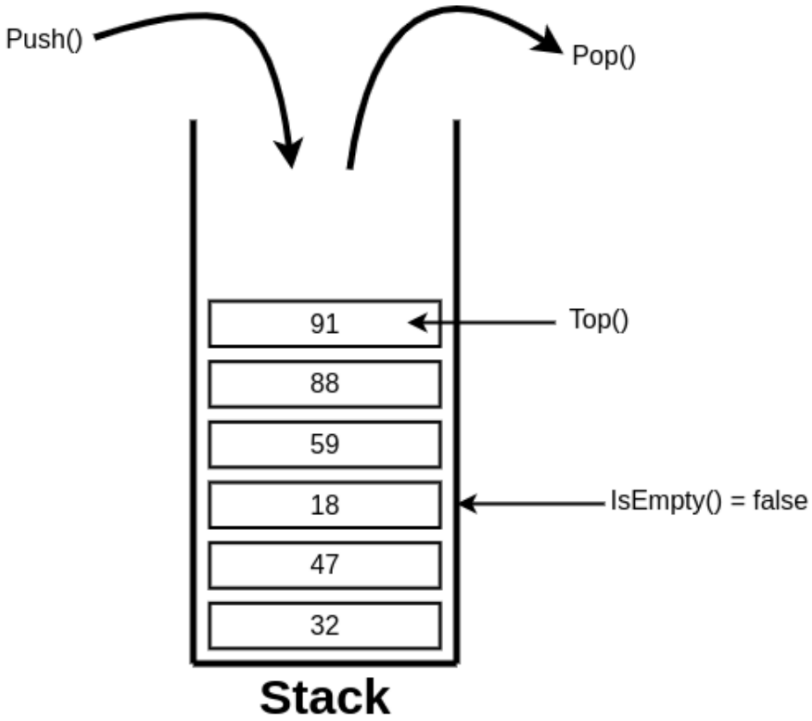
2. Undo Mechanism in Text Editors: When you press 'Ctrl+Z' to undo your last action in a text editor, the program uses a stack to keep track of all the actions you perform. Each time you perform an action, it gets pushed onto the stack, and when you undo an action, it gets popped off the stack.

3. Expression Conversion (Infix to Postfix/Prefix): Algorithms for converting expressions from infix notation (e.g., A + B) to postfix (e.g., AB+) or prefix notation (e.g., +AB) use stacks to manage operators and operands during the conversion process.

4. Recursive Algorithms: Many algorithms, particularly those involving recursion, use stacks to keep track of recursive function calls.

5. Function Call Management in Programming Languages: When a function calls another function, a stack is used to keep track of the return points and local variables of each function. This is known as the call stack.

Diagram:



Explain Stack ADT:

The Stack Abstract Data Type (ADT) is a collection of elements with two primary operations: push and pop. The stack follows the Last-In-First-Out (LIFO) principle, meaning the last element added (pushed) to the stack is the first one to be removed (popped) and does not allow random access to the elements of the stack

Key Operations of Stack ADT

1. Push: Adds an element to the top of the stack

2. Pop: Removes and returns the top element of the stack

3. Peek: Returns the top element without removing it

4. isEmpty: Checks if the stack is empty, and displays a stack underflow error

5. isFull: Checks if the stack is full, and displays a stack overflow error

6. display: Displays elements present in the stack at that moment

Algorithm for creation, insertion, deletion, displaying an element in stack [static implementation]:

1. Creation (Initialization)

Define a Stack Structure:

- An array a of size n to hold stack elements (n=5)
- An integer top to track the index of the top element

Initialize the Stack:

• Set top to -1 to indicate that the stack is empty

2. Insertion (Push Operation)

Check if the Stack is Full:

- Compare top with n-1. If top equals n-1, the stack is full

Add an Element to the Stack:

- Increment the top index
- Store the new element at the position top in the array

3. Deletion (Pop Operation)

Check if the Stack is Empty:

- Compare top with -1. If top equals -1, the stack is empty

Remove an Element from the Stack:

- Decrement the top index to remove the top element

4. Displaying (Display Operation)

Check if the Stack is Empty:

- Compare top with -1. If top equals -1, the stack is empty

5. Display Stack Elements:

- Iterate from the top index to 0
- Print each element from the array

Program source code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10

typedef struct Stack{
    int data[MAX_SIZE];
    int top;
} Stack;

Stack createStack() { // Function to create a stack
    Stack newStack;
    newStack.top = -1;
    return newStack;
}

int isFull(Stack *stack) { // Function to check if stack is full
    return stack->top == MAX_SIZE - 1;
}

int isEmpty(Stack *stack) { // Function to check if stack is empty
    return stack->top == -1;
}

void push(Stack *stack, int data) { // Function to push data into stack &
increase top by 1
    if (isFull(stack)) {
        printf("Stack is full\n");
        return;
    }
    stack->data[++stack->top] = data;
}

int pop(Stack* stack) { // Function to pop data from stack & decrease top
by 1
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
```

```
        return -1;
    }
    return stack->data[stack->top--];
}

void display(Stack* stack) { // Function to display stack elements
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return;
    }
    for (int i = stack->top; i >= 0; i--) {
        printf("%d ", stack->data[i]);
    }
    printf("\n");
}

int peek(Stack* stack) { // Function to display top element of stack
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return -1;
    }
    printf("%d\n", stack->data[stack->top]);
}

int main() {
    Stack stack = createStack();
    int choice, data;

    while (1) {
        printf("Stack Operations\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Peek\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data: ");
                scanf("%d", &data);
                push(&stack, data);
                break;

            case 2:
```

```
                data = pop(&stack);
                if (data != -1) {
                    printf("Popped element: %d\n", data);
                }
                break;

            case 3:
                display(&stack);
                break;

            case 4:
                data = peek(&stack);
                peek(&stack);
                break;

            case 5:
                printf("Exiting\n");
                exit(0);
                break;

            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

Output Screenshots:

```
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 2
Popped element: 4
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 3
6 2
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter data: 9
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 4
9
9
```

```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs\" ; if
 ($?) { gcc stack.c -o stack } ; if ($?) { .\stack }
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter data: 2
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter data: 6
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1
Enter data: 4
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 3
4 6 2
```

```
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 4
9
9
Stack Operations
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 5
Exiting
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs>
```

PostLab Questions:

Q1) List 5 Applications of Stack Data Structures.

1) **Evaluation of Arithmetic Expressions** - Algorithms for converting expressions from infix notation (e.g., A + B) to postfix (e.g., AB+) or prefix notation (e.g., +AB) use stacks to manage operators and operands during the conversion process.
2) **Backtracking** - Backtracking is another application of Stack. It is a recursive algorithm that is used for solving the optimization problem.
3) **Reversing data** - To reverse a given set of data, we need to reorder the data so that the first and last elements are exchanged, and so on. Hence, we use stacks because of the LIFO principle.
4) **Processing Function calls -** Stack plays an important role in programs that call several functions in succession.
5) **Delimiter checking/Parsing** - The common application of Stack is delimiter checking, i.e., parsing that involves analysing a source program syntactically.

2) Write a program to implement stack using array to reverse the text given by the user. (The program should be implement using stack array representation. The program should repeatedly add each character of the string into the stack. After all additions, all characters are popped from the stack and concatenate to form the string and displayed.)
Input- Hello
Output- olleH

Ans:

```c
#include <stdio.h>
#include <string.h>

#define MAX 100

typedef struct
{
    int top;
    char items[MAX];
}Stack;

void initialize(Stack *s)
{
    s->top = -1;
}


int isFull(Stack *s)
{
    return s->top == MAX - 1;
}

int isEmpty(Stack *s)
{
    return s->top == -1;
}

void push(Stack *s, char item)
{
    if (isFull(s))
    {
        printf("Stack is full.\n");
        return;
    }
```

```
    s->items[++(s->top)] = item;
}


char pop(Stack *s)
{
    if (isEmpty(s))
    {
        printf("Stack is empty.\n");
        return '\0';
    }
    return s->items[(s->top)--];
}


void reverseString(char str[])
{
    int n = strlen(str);
    Stack s;
    initialize(&s);

    for (int i = 0; i < n; i++)
    {
        push(&s, str[i]);
    }

    for (int i = 0; i < n; i++)
    {
        str[i] = pop(&s);
    }
}

int main()
{
    char str[MAX];

    printf("Enter a string: ");
    scanf("%s", str);

    reverseString(str);

    printf("Reversed string: %s\n", str);

}
```

```
/tmp/WHufVziuUg.o
Enter a string: Hello
Reversed string: olleH


=== Code Execution Successful ===
```

```
/tmp/MXkJbACAUM.o
Enter a string: whoa
Reversed string: aohw


=== Code Execution Successful ===
```

3) Write a program to convert a given non-negative decimal integer into its binary representation using a stack.

(The program should implement the stack using an array and perform the conversion by repeatedly dividing the number by 2 and pushing the remainders onto the stack. After all divisions are done, pop the elements from the stack to form the binary representation)

Input: 10

Output: 1010

Input: 4

Output: 100

Ans:

```c
#include <stdio.h>
#include <string.h>
```

```c
#define MAX 100

typedef struct
{
    int top;
    char items[MAX];
}Stack;

void initialize(Stack *s)
{
    s->top = -1;
}


int isFull(Stack *s)
{
    return s->top == MAX - 1;
}

int isEmpty(Stack *s)
{
    return s->top == -1;
}

void push(Stack *s, char item)
{
    if (isFull(s))
    {
        printf("Stack is full.\n");
        return;
    }
    s->items[++(s->top)] = item;
}


char pop(Stack *s)
{
    if (isEmpty(s))
    {
        printf("Stack is empty.\n");
        return '\0';
    }
    return s->items[(s->top)--];
}
```

```c
void decimalToBinary(int n)
{
    Stack s;
    initialize(&s);

    if (n == 0)
    {
        printf("Binary representation: 0\n");
        return;
    }


    while (n > 0)
    {
        int remainder = n % 2;
        push(&s, remainder);
        n /= 2;
    }


    printf("Binary representation: ");
    while (!isEmpty(&s))
    {
        printf("%d", pop(&s));
    }
    printf("\n");
}

int main()
{
    int number;

    printf("Enter a non-negative integer: ");
    scanf("%d", &number);

    decimalToBinary(number);

    return 0;
}
```

```
/tmp/U80QufWXCx.o
Enter a non-negative integer: 10
Binary representation: 1010


=== Code Execution Successful ===
```

```
/tmp/v4MFYckDv9.o
Enter a non-negative integer: 4
Binary representation: 100


=== Code Execution Successful ===
```