**Aim:** To write a C program that implements **push** and **pop** operations on a stack using a linear array.

**Apparatus / Software Required:**

1. Computer System
2. GCC Compiler / Turbo C
3. Text Editor or IDE (Code::Blocks, Dev-C++, VS Code, etc.)

**Theory:**

A **stack** is a linear data structure that follows the **LIFO (Last In First Out)** principle.

1. **Push Operation:** Adds an element on the **top** of the stack.
2. **Pop Operation:** Removes the element from the **top** of the stack.

We maintain a variable **top** that keeps track of the current top index of the stack.

1. Initially, top = -1 (stack is empty).
2. On **Push**, increment top and insert element.
3. On **Pop**, remove element at top and decrement top.

**Overflow condition:** When stack is full and a new element cannot be pushed.  
**Underflow condition:** When stack is empty and a pop operation is attempted.

**Steps:**

1. Define an array to implement stack and a variable top.
2. Initialize top = -1.
3. For **Push operation**:
   * Check if top == MAX-1 → Stack Overflow.
   * Else increment top and insert element at stack[top].
4. For **Pop operation**:
   * Check if top == -1 → Stack Underflow.
   * Else delete element from stack[top] and decrement top.
5. Provide menu-driven options for user to perform Push, Pop, and Display operations.

**Advantages:**

1. Stack operations (Push/Pop) are **constant time (O(1))**.
2. Useful in **expression evaluation, recursion, undo mechanisms** etc.

**Limitations:**

1. Fixed size in array implementation (may cause **overflow**).
2. Less flexible compared to linked list implementation.

**Algorithm:**

1. Start
2. Initialize top = -1
3. Repeat until exit:
   * Display menu: Push / Pop / Display / Exit
   * If Push: check for overflow, else insert element
   * If Pop: check for underflow, else remove element
   * If Display: print all elements from top to 0
4. Stop

**Program:**

#include <stdio.h>

#define MAX 5 // Maximum size of stack

int stack[MAX];

int top = -1; // Initially stack is empty

// Function to push element into stack

void push(int element) {

if(top == MAX - 1) {

printf("Stack Overflow! Cannot push %d\n", element);

} else {

top++;

stack[top] = element;

printf("%d pushed into stack.\n", element);

}

}

// Function to pop element from stack

void pop() {

if(top == -1) {

printf("Stack Underflow! No element to pop.\n");

} else {

printf("%d popped from stack.\n", stack[top]);

top--;

}

}

// Function to display stack

void display() {

if(top == -1) {

printf("Stack is empty.\n");

} else {

printf("Stack elements are:\n");

for(int i = top; i >= 0; i--) {

printf("%d ", stack[i]);

}

printf("\n");

}

}

int main() {

int choice, element;

while(1) {

printf("\n--- Stack Menu ---\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1:

printf("Enter element to push: ");

scanf("%d", &element);

push(element);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

return 0; // Exit program

default:

printf("Invalid choice! Try again.\n");

}

}

}

**Sample Output:**

--- Stack Menu ---

1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 1

Enter element to push: 10

10 pushed into stack.

--- Stack Menu ---

Enter your choice: 1

Enter element to push: 20

20 pushed into stack.

--- Stack Menu ---

Enter your choice: 3

Stack elements are:

20 10

--- Stack Menu ---

Enter your choice: 2

20 popped from stack.

--- Stack Menu ---

Enter your choice: 2

10 popped from stack.

--- Stack Menu ---

Enter your choice: 2

Stack Underflow! No element to pop.

**Conclusion:**

The program successfully implements **stack operations (Push and Pop)** using a linear array. The stack follows the **LIFO principle**, with overflow and underflow conditions properly handled.