Practical-1

Aim : Write a program for implementing a MINSTACK which should support operations like push, pop, overflow, underflow, display

a.Construct a stack of N-capacity

b. Push elements

c. Pop elements

d. Top element

e. Retrieve the min element from the stack

Algorithm:-

1. Initialization:

- Create a MinStack instance minStack with a capacity of 5.

2. Push Operation:

- push(int element)

o Check if the stack is not full.

Push the element onto the stack.

o If minStack is empty or the element is less than or equal to the current minimum, push element onto minStack.

3. Pop Operation:

- pop()

o Check if the stack is not empty.

o Pop the top element from the stack.

o If the popped element is the current minimum, pop from minStack as well.

4. Top Operation:

- top()

o Check if the stack is not empty.

o Return the top element of the stack.

5. GetMin Operation:

- getMin()

o Check if minStack is not empty.

o Return the minimum element from minStack.

6. Display Operation:

- display()

o Print "Elements in the stack:".

o Iterate over elements in the stack and print each element.

7. Main Method Execution:

- Instantiate a MinStack object with a capacity of 5.
- Push elements (3, 5, 2, 7, 1) to the stack.
- Display elements in the stack.
- Print the minimum element in the stack.
- Pop two elements from the stack.
- Print the top element in the stack.

Example:-

Step 1: Push 3

- **Operation:** Push 3 onto the stack.
- **State After Operation:**
    - **top = 0**
    - **data = [3]**
    - **minData = [3]** (since 3 is the only element, it is also the minimum)

Step 2: Push 5

- **Operation:** Push 5 onto the stack.
- **State After Operation:**
    - **top = 1**
    - **data = [3, 5]**

    - **minData = [3, 3]** (the minimum remains 3, as 5 is greater than 3)

Step 3: Push 2

- **Operation:** Push 2 onto the stack.
- **State After Operation:**
    - **top = 2**
    - **data = [3, 5, 2]**
    - **minData = [3, 3, 2]** (2 is less than the current minimum, so it becomes the new minimum)

Step 4: Push 7

- **Operation:** Push 7 onto the stack.
- **State After Operation:**
    - **top = 3**
    - **data = [3, 5, 2, 7]**
    - **minData = [3, 3, 2, 2]** (the minimum remains 2, as 7 is greater than 2)

Step 5: Push 1

- **Operation:** Push 1 onto the stack.
- **State After Operation:**
    - **top = 4**
    - **data = [3, 5, 2, 7, 1]**
    - **minData = [3, 3, 2, 2, 1]** (1 is less than the current minimum, so it becomes the new minimum)

Current State of the Stack

- **Stack Elements:** [3, 5, 2, 7, 1]
- **Minimum Elements:** [3, 3, 2, 2, 1]
- **Top Index:** 4

Step 6: Display Stack

- **Output:** "Elements in the stack: 3 5 2 7 1"

Step 7: Get Minimum Element

- **Operation:** Retrieve the minimum element.
- **Output:** "Min element in stack: 1"

Step 8: Pop (Remove Top Element)

- **Operation:** Pop the top element (1).
- **State After Operation:**
    - **top = 3** (decremented)
    - **data = [3, 5, 2, 7]**
    - **minData = [3, 3, 2, 2]** (1 was the minimum, but now the minimum is still 2)

Step 9: Pop (Remove Next Top Element)

- **Operation:** Pop the next top element (7).
- **State After Operation:**
    - **top = 2** (decremented)
    - **data = [3, 5, 2]**
    - **minData = [3, 3, 2]** (the minimum remains 2)

Step 10: Get Top Element

- **Operation:** Retrieve the top element.
- **Output:** "Top element in stack: 2"

Final State of the Stack

- **Stack Elements:** [3, 5, 2]
- **Minimum Elements:** [3, 3, 2]
- **Top Index:** 2

Program:-

Output:-

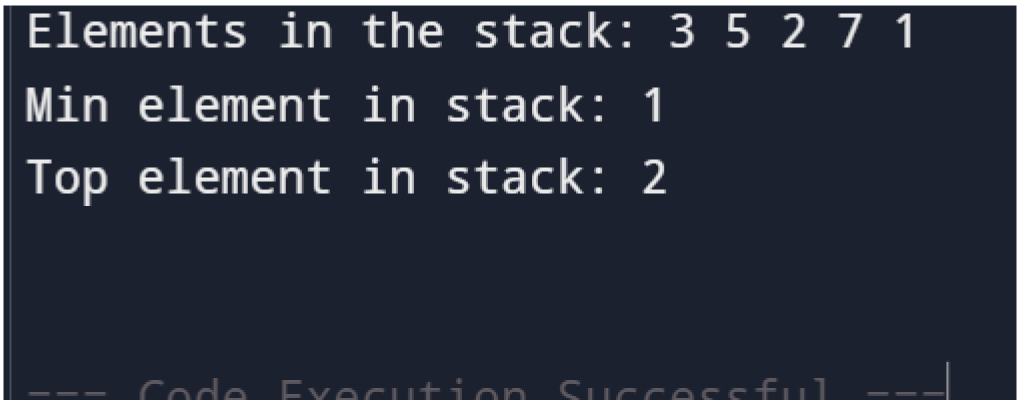
```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>   // define various types of limits and constants

#define MAX_SIZE 5 // Define the maximum capacity of the stack

// typedef struct that represents a stack with additional functionality to track the
minimum element efficiently
```

```
typedef struct
 {
    int data[MAX_SIZE];      // Array to hold stack elements
    int minData[MAX_SIZE];   // Array to hold minimum elements
    int top;                 // Index of the top element
} MinStack;   // keep recorrd of min element in satck


// Function to initialize the stack
void initStack(MinStack *stack) {
    stack->top = -1; // Stack is initially empty
// stack->top refers to the top member of the MinStack
}


// Function to check if the stack is full
int isFull(MinStack *stack) //passing a pointer, the function can access the members of
the MinStack
 {
    return stack->top == MAX_SIZE - 1;  //stack->top refers to the top member of
the MinStack  and checks if the top index is equal to MAX_SIZE - 1
}


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


// Function to push an element onto the stack
void push(MinStack *stack, int element) {
    if (isFull(stack)) {
        printf("Overflow\n");
        return;
    }
    stack->top++;
    stack->data[stack->top] = element;  //top member of the MinStack structure, which
keeps track of the index

    // Update the min stack
//also check if top element of min stack is is less than or equal to the current minimum
element
    if (stack->top == 0 || element <= stack->minData[stack->top - 1]) {
        stack->minData[stack->top] = element;
    } else {
        stack->minData[stack->top] = stack->minData[stack->top - 1];
    }
}

// Function to pop an element from the stack
void pop(MinStack *stack) {
    if (isEmpty(stack)) {
        printf("Underflow\n");
        return;
    }
    int popped = stack->data[stack->top];
    stack->top--;
    if (popped == stack->minData[stack->top + 1]) {
        stack->minData[stack->top + 1] = INT_MAX; // Reset min if necessary
    }
}

// Function to get the top element of the stack
```

```
int top(MinStack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return -1;
    }
    return stack->data[stack->top];
}

// Function to get the minimum element from the stack
int getMin(MinStack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return -1;
    }
    return stack->minData[stack->top];
}

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

```
// Main function to demonstrate the MinStack
int main() {
    MinStack minStack;
    initStack(&minStack);

    push(&minStack, 3);
    push(&minStack, 5);
    push(&minStack, 2);
    push(&minStack, 7);
    push(&minStack, 1);

    display(&minStack);
    printf("Min element in stack: %d\n", getMin(&minStack));

    pop(&minStack);
    pop(&minStack);

    printf("Top element in stack: %d\n", top(&minStack));

    return 0;
}
```

Output:-



```
Elements in the stack: 3 5 2 7 1
Min element in stack: 1
Top element in stack: 2



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