

# Rajalakshmi Engineering College

Name: shobbika T  
Email: 240701502@rajalakshmi.edu.in  
Roll no: 240701502  
Phone: 7305423247  
Branch: REC  
Department: I CSE FE  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## NeoColab\_REC\_CS23231\_DATA STRUCTURES

### REC\_DS using C\_Week 3\_CY

Attempt : 1  
Total Mark : 30  
Marks Obtained : 30

### Section 1 : Coding

#### 1. Problem Statement

You are required to implement a stack data structure using a singly linked list that follows the Last In, First Out (LIFO) principle.

The stack should support the following operations: push, pop, display, and peek.

#### *Input Format*

The input consists of four space-separated integers N, representing the elements to be pushed onto the stack.

#### *Output Format*

The first line of output displays all four elements in a single line separated by a space.

The second line of output is left blank to indicate the pop operation without displaying anything.

The third line of output displays the space separated stack elements in the same line after the pop operation.

The fourth line of output displays the top element of the stack using the peek operation.

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 11 22 33 44

Output: 44 33 22 11

33 22 11

33

### **Answer**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* top = NULL;
void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
}
void pop() {
    if (top == NULL) return;
    struct Node* temp = top;
    top = top->next;
    free(temp);
}
int peek() {
```

```

    if (top == NULL) return -1;
    return top->data;
}

void display() {
    struct Node* temp = top;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

int main() {
    int a, b, c, d;
    scanf("%d %d %d %d", &a, &b, &c, &d);
    push(a);
    push(b);
    push(c);
    push(d);
    display();
    printf("\n");
    pop();
    printf("\n");
    display();
    printf("\n");
    printf("%d\n", peek());

    return 0;
}

```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Buvi is working on a project that requires implementing an array-stack data structure with an additional feature to find the minimum element.

Buvi needs to implement a program that simulates a stack with the following functionalities:

**Push:** Adds an element onto the stack. **Pop:** Removes the top element from the stack. **Find Minimum:** Finds the minimum element in the stack.

Buvi's implementation should efficiently handle these operations with a maximum stack size of 20.

### ***Input Format***

The first line of input consists of an integer N, representing the number of elements to push onto the stack.

The second line consists of N space-separated integer values, representing the elements to be pushed onto the stack.

### ***Output Format***

The first line of output displays "Minimum element in the stack: " followed by the minimum element in the stack after pushing all elements.

The second line displays "Popped element: " followed by the popped element.

The third line displays "Minimum element in the stack after popping: " followed by the minimum element in the stack after popping one element.

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 4

5 2 8 1

Output: Minimum element in the stack: 1

Popped element: 1

Minimum element in the stack after popping: 2

### ***Answer***

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

```
#include <limits.h>
```

```
#define MAX 20
```

```
int stack[MAX];
```

```
int minStack[MAX];
```

```
int top = -1;
```

```

int minTop = -1;
void push(int value) {
    if (top >= MAX - 1) {
        printf("Stack overflow\n");
        return;
    }
    stack[++top] = value;
    if (minTop == -1 || value <= minStack[minTop]) {
        minStack[++minTop] = value;
    }
}

```

```

int pop() {
    if (top == -1) {
        printf("Stack underflow\n");
        return -1;
    }

```

```

    int popped = stack[top--];

```

```

    if (popped == minStack[minTop]) {
        minTop--;
    }

```

```

    return popped;
}
int getMin() {
    if (minTop == -1) return INT_MAX;
    return minStack[minTop];
}

```

```

int main() {
    int N, i, value;
    scanf("%d", &N);

```

```

    for (i = 0; i < N; i++) {
        scanf("%d", &value);
        push(value);
    }

```

```

    printf("Minimum element in the stack: %d\n", getMin());

```

```

    int poppedElement = pop();

```

```
printf("Popped element: %d\n", poppedElement);  
printf("Minimum element in the stack after popping: %d\n", getMin());  
return 0;  
}
```

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

In an educational setting, Professor Smith tasks Computer Science students with designing an algorithm to evaluate postfix expressions efficiently, fostering problem-solving skills and understanding of stack-based computations.

The program prompts users to input a postfix expression, evaluates it, and displays the result, aiding students in honing their coding abilities.

#### ***Input Format***

The input consists of the postfix mathematical expression.

The expression will contain real numbers and mathematical operators ( +, -, \*, / ), without any space.

#### ***Output Format***

The output prints the result of evaluating the given postfix expression.

Refer to the sample output for formatting specifications.

#### ***Sample Test Case***

Input: 82/

Output: 4

#### ***Answer***

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

```

#include <ctype.h>
#include <stdlib.h>

#define MAX 100

float stack[MAX];
int top = -1;
void push(float value) {
    if (top >= MAX - 1) {
        printf("Stack overflow\n");
        return;
    }
    stack[++top] = value;
}
float pop() {
    if (top == -1) {
        printf("Stack underflow\n");
        return 0;
    }
    return stack[top--];
}
float evaluatePostfix(const char* expr) {
    for (int i = 0; expr[i] != '\0'; i++) {
        char ch = expr[i];

        if (isdigit(ch)) {
            push((float)(ch - '0'));
        } else {
            float val2 = pop();
            float val1 = pop();
            switch (ch) {
                case '+': push(val1 + val2); break;
                case '-': push(val1 - val2); break;
                case '*': push(val1 * val2); break;
                case '/': push(val1 / val2); break;
                default:
                    printf("Invalid operator: %c\n", ch);
                    return 0;
            }
        }
    }
}

```

```
        return pop();
    }

    int main() {
        char expr[100];
        scanf("%s", expr);

        float result = evaluatePostfix(expr);
        if ((int)result == result)
            printf("%.0f\n", result);
        else
            printf("%f\n", result);

        return 0;
    }
```

**Status :** Correct

**Marks :** 10/10