Rajalakshmi Engineering College

Name: Jeffery Antony J
Email: 241901041@rajalakshmi.edu.in
Roll no: 241901041
Phone: 7305663808
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};


void push(struct Node** top, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *top;
    *top = newNode;
}


void pop(struct Node** top) {
    if (*top != NULL) {
```

```
        struct Node* temp = *top;
        *top = (*top)->next;
        free(temp);
    }
}


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


void peek(struct Node* top) {
    if (top != NULL) {
        printf("%d\n", top->data);
    }
}

int main() {
    struct Node* top = NULL;
    int a, b, c, d;


    scanf("%d %d %d %d", &a, &b, &c, &d);

    push(&top, a);
    push(&top, b);
    push(&top, c);
    push(&top, d);

    display(top);

    pop(&top);


    printf("\n");
```

```
    display(top);

    peek(top);

    return 0;
}
```

Status : Correct ***Marks : 10/10***

2. 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

```c
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define MAX 100

float stack[MAX];
int top = -1;


void push(float value) {
    if (top < MAX - 1) {
        stack[++top] = value;
    }
}


float pop() {
    if (top >= 0) {
        return stack[top--];
    }
    return 0;
}


int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}


float evaluatePostfix(const char* expr) {
    int i = 0;
    while (expr[i] != '\0') {
        if (isdigit(expr[i])) {

            push(expr[i] - '0');
        } else if (isOperator(expr[i])) {
            float val2 = pop();
            float val1 = pop();
            switch (expr[i]) {
                case '+': push(val1 + val2); break;
```

```
            case '-': push(val1 - val2); break;
            case '*': push(val1 * val2); break;
            case '/': push(val1 / val2); break;
        }
      }
      i++;
   }
   return pop();
}

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

   float result = evaluatePostfix(expr);


   if (result == (int)result)
      printf("%d\n", (int)result);
   else
      printf("%f\n", result);

   return 0;
}
```

Status : Correct **Marks : 10/10**

3. 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>

#define MAX 20

int stack[MAX];
int minStack[MAX];
int top = -1;
int minTop = -1;

void push(int value) {
```

```
    if (top < MAX - 1) {
        stack[++top] = value;
        if (minTop == -1 || value <= minStack[minTop]) {
            minStack[++minTop] = value;
        }
    }
}


int pop() {
    if (top == -1)
        return -1;
    int popped = stack[top--];
    if (popped == minStack[minTop]) {
        minTop--;
    }
    return popped;
}


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

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

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

    int popped = pop();
    printf("Popped element: %d\n", popped);

    printf("Minimum element in the stack after popping: %d", getMin());

    return 0;
```

```
}
```

Status : Correct *Marks : 10/10*