Rajalakshmi Engineering College

Name: SHIVANISREE K B
Email: 240701501@rajalakshmi.edu.in
Roll no: 240701501
Phone: 7358464804
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

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

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

#define MAX 100

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

// Push function
void push(float val) {
    if (top >= MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = val;
}

// Pop function
float pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        exit(1);
    }
    return stack[top--];
}
```

```
// Main function to evaluate postfix expression
int main() {
    char expr[MAX];
    scanf("%s", expr);  // Read postfix expression as string

    for (int i = 0; expr[i] != '\0'; i++) {
        char ch = expr[i];

        if (isdigit(ch)) {
            // Convert char digit to float and push
            push((float)(ch - '0'));
        } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            float val2 = pop();
            float val1 = pop();
            float result;

            switch (ch) {
                case '+': result = val1 + val2; break;
                case '-': result = val1 - val2; break;
                case '*': result = val1 * val2; break;
                case '/': result = val1 / val2; break;
            }

            push(result);
        }
    }

    // Final result
    float finalResult = pop();

    // If result is integer, print as integer
    if (finalResult == (int)finalResult)
        printf("%d\n", (int)finalResult);
    else
        printf("%g\n", finalResult);

    return 0;
}
```

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

2. Problem Statement

Suppose you are building a calculator application that allows users to enter mathematical expressions in infix notation. One of the key features of your calculator is the ability to convert the entered expression to postfix notation using a Stack data structure.

Write a function to convert infix notation to postfix notation using a Stack.

Input Format

The input consists of a string, an infix expression that includes only digits(0-9), and operators(+, -, *, /).

Output Format

The output displays the equivalent postfix expression of the given infix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1+2*3/4-5

Output: 123*4/+5-

Answer

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

#define MAX 100

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

// Push onto stack
void push(char ch) {
```

```
    if (top >= MAX - 1) return;
    stack[++top] = ch;
}

// Pop from stack
char pop() {
    if (top == -1) return -1;
    return stack[top--];
}

// Peek top of stack
char peek() {
    if (top == -1) return -1;
    return stack[top];
}

// Check precedence of operators
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}

// Convert infix to postfix
void infixToPostfix(char* infix) {
    char postfix[MAX];
    int k = 0;

    for (int i = 0; infix[i] != '\0'; i++) {
        char ch = infix[i];

        if (isdigit(ch)) {
            postfix[k++] = ch;  // Append operands directly
        }
        else if (ch == '(') {
            push(ch);
        }
        else if (ch == ')') {
            while (peek() != '(') {
                postfix[k++] = pop();
            }
            pop();  // Discard '('
```

```
        }
        else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            while (top != -1 && precedence(peek()) >= precedence(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }

    // Pop remaining operators
    while (top != -1) {
        postfix[k++] = pop();
    }

    postfix[k] = '\0';
    printf("%s\n", postfix);
}

// Main
int main() {
    char infix[MAX];
    scanf("%s", infix);
    infixToPostfix(infix);
    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

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

#define MAX 20

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

// Push element onto both main and min stack
void push(int value) {
    if (top == MAX - 1) return; // Stack full
```

```
    stack[++top] = value;

    if (minTop == -1 || value <= minStack[minTop])
        minStack[++minTop] = value;
}

// Pop element from both main and min stack
int pop() {
    if (top == -1) return -1; // Stack empty
    int popped = stack[top--];
    if (popped == minStack[minTop])
        minTop--;
    return popped;
}

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

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

    for (int 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\n", getMin());

    return 0;
}
```

Status : Correct *Marks : 10/10*