Rajalakshmi Engineering College

Name: Sowmyalakshmi   N
Email: 240701524@rajalakshmi.edu.in
Roll no: 240701524
Phone: 9003767185
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

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>
#define MAX 20

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

// Push operation
void push(int value) {
    if (top < MAX - 1) {
        stack[++top] = value;

        // Push to minStack if it's empty or value <= current min
        if (minTop == -1 || value <= minStack[minTop]) {
            minStack[++minTop] = value;
        }
    }
}

// Pop operation
int pop() {
    if (top == -1)
        return -1; // Stack underflow

    int popped = stack[top--];

    // Pop from minStack if necessary
    if (popped == minStack[minTop]) {
        minTop--;
    }

    return popped;
}

// Get current minimum
int getMin() {
    if (minTop != -1)
        return minStack[minTop];
    return -1; // Empty stack
}

int main() {
    int N, i, value;

    // Read number of elements
    scanf("%d", &N);

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

    // Output: minimum after all pushes
    printf("Minimum element in the stack: %d\n", getMin());

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

    // Output: new minimum after pop
    printf("Minimum element in the stack after popping: %d\n", getMin());

    return 0;
}
```

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

2. 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>

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

// Stack top pointer
struct Node* top = NULL;

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

// Pop operation
void pop() {
   if (top != NULL) {
      struct Node* temp = top;
      top = top->next;
      free(temp);
   }
}

// Peek operation
int peek() {
   if (top != NULL) {
        return top->data;
    }
    return -1; // Return -1 if stack is empty (can be modified)
}

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

// Main function
int main() {
    int a, b, c, d;

    // Input: read 4 space-separated integers
    scanf("%d %d %d %d", &a, &b, &c, &d);

    // Push elements to stack in order
    push(a);
    push(b);
    push(c);
    push(d);

    // Output after all pushes
    display();

    // Blank line for pop operation
    pop();
    printf("\n");

    // Output after one pop
    display();

    // Output top element
    printf("%d\n", peek());

    return 0;
```

```
}
```

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

3. 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
```

```
// Stack structure
char stack[MAX];
int top = -1;

// Function to push an element onto the stack
void push(char c) {
    if (top < MAX - 1) {
        stack[++top] = c;
    }
}

// Function to pop an element from the stack
char pop() {
    if (top != -1)
        return stack[top--];
    return '\0';
}

// Function to peek the top element of the stack
char peek() {
    if (top != -1)
        return stack[top];
    return '\0';
}

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

// Function to check if character is an operator
int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}

// Infix to Postfix conversion function
void infixToPostfix(char* infix, char* postfix) {
    int i, j = 0;
    char ch;
    for (i = 0; infix[i] != '\0'; i++) {
        ch = infix[i];

        if (isdigit(ch)) {
            postfix[j++] = ch;
        } else if (ch == '(') {
            push(ch);
        } else if (ch == ')') {
            while (top != -1 && peek() != '(') {
                postfix[j++] = pop();
            }
            pop(); // remove '(' from stack
        } else if (isOperator(ch)) {
            while (top != -1 && precedence(peek()) >= precedence(ch)) {
                postfix[j++] = pop();
            }
            push(ch);
        }
    }

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

    postfix[j] = '\0'; // null-terminate the postfix expression
}

// Main function
int main() {
    char infix[31], postfix[31];

    // Input
    //printf("Enter infix expression: ");
    scanf("%30s", infix);  // Read up to 30 characters

    // Conversion
    infixToPostfix(infix, postfix);

    // Output
    printf("%s\n", postfix);
    return 0;
}
```

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