Rajalakshmi Engineering College

Name: Lakshmi Narayanan S
Email: 241801133@rajalakshmi.edu.in
Roll no: 241801133
Phone: 9345832054
Branch: REC
Department: I AI & DS - AE
Batch: 2028
Degree: B.E - AI & DS

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;
};

struct Stack {
    struct Node* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

void push(struct Stack* stack, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
```

```
    newNode->next = stack->top;
    stack->top = newNode;
}

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

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

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

int main() {
    struct Stack stack;
    initStack(&stack);

    // Read input
    int values[4];
    for (int i = 0; i < 4; i++) {
        scanf("%d", &values[i]);
    }

    // Push elements onto the stack
    for (int i = 0; i < 4; i++) {
        push(&stack, values[i]);
    }
```

```
    // Display stack after push
    display(&stack);

    // Pop operation (with blank line in output)
    pop(&stack);
    printf("\n");

    // Display stack after pop
    display(&stack);

    // Peek the top element
    peek(&stack);

    return 0;
}
```

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

2. Problem Statement

Latha is taking a computer science course and has recently learned about infix and postfix expressions. She is fascinated by the idea of converting infix expressions into postfix notation. To practice this concept, she wants to implement a program that can perform the conversion for her.

Help Latha by designing a program that takes an infix expression as input and outputs its equivalent postfix notation.

Example

Input:

(3+4)5

Output:

34+5

Input Format

The input consists of a string, the infix expression to be converted to postfix notation.

Output Format

The output displays a string, the postfix expression equivalent of the input infix expression.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: A+B*C-D/E

Output: ABC*+DE/-

Answer

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

// Stack structure definition
#define MAX 100
char stack[MAX];
int top = -1;

// Function to get the precedence of operators
int precedence(char op) {
    if (op == '+' || op == '-') {
        return 1;
    } else if (op == '*' || op == '/') {
        return 2;
    } else if (op == '^') {
        return 3;  // '^' has higher precedence
    }
    return 0;
}

// Function to check if a character is an operand (either a number or a variable)
int isOperand(char ch) {
    return isalpha(ch) || isdigit(ch);
}
```

```
// Function to perform the infix to postfix conversion
void infixToPostfix(char* infix) {
    char postfix[MAX];
    int j = 0;  // Index for postfix expression

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

        // If current character is an operand, add it to the postfix expression
        if (isOperand(current)) {
            postfix[j++] = current;
        }
        // If current character is '(', push it to the stack
        else if (current == '(') {
            stack[++top] = current;
        }
        // If current character is ')', pop from stack to postfix until '(' is encountered
        else if (current == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = stack[top--];
            }
            top--;  // Pop '(' from the stack
        }
        // If current character is an operator
        else if (current == '+' || current == '-' || current == '*' || current == '/' || current ==
'^') {
            while (top != -1 && precedence(stack[top]) >= precedence(current)) {
                postfix[j++] = stack[top--];
            }
            stack[++top] = current;
        }
    }

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

    postfix[j] = '\0';  // Null terminate the postfix expression
    printf("%s\n", postfix);
}
```

```
int main() {
    // Read input infix expression
    char infix[MAX];
    fgets(infix, MAX, stdin);
    infix[strcspn(infix, "\n")] = 0;  // Remove newline character from input if present

    // Convert infix to postfix and output the result
    infixToPostfix(infix);

    return 0;
}
```

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

3. Problem Statement

Siri is a computer science student who loves solving mathematical problems. She recently learned about infix and postfix expressions and was fascinated by how they can be used to evaluate mathematical expressions.

She decided to write a program to convert an infix expression with operators to its postfix form. Help Siri in writing the program.

Input Format

The input consists of a single line containing an infix expression.

Output Format

The output prints a single line containing the postfix expression equivalent to the given infix expression.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: (2 + 3) * 4

Output: 23+4*

Answer

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

#define MAX 50
char stack[MAX];
int top = -1;

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

// Function to check if the character is an operand (digit)
int isOperand(char ch) {
    return isdigit(ch);  // Check if the character is a digit
}

// Function to perform infix to postfix conversion
void infixToPostfix(char* infix) {
    char postfix[MAX];
    int j = 0;  // Index for postfix expression

    // Remove spaces from the infix expression
    int len = strlen(infix);
    for (int i = 0; i < len; i++) {
        if (infix[i] == ' ') {
            // Skip spaces
            continue;
        }

        char current = infix[i];
```

```
        // If current character is an operand, add it to the postfix expression
        if (isOperand(current)) {
            postfix[j++] = current;
        }
        // If current character is '(', push it to the stack
        else if (current == '(') {
            stack[++top] = current;
        }
        // If current character is ')', pop from stack to postfix until '(' is encountered
        else if (current == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = stack[top--];
            }
            top--;  // Pop '(' from the stack
        }
        // If current character is an operator
        else if (current == '+' || current == '-' || current == '*' || current == '/') {
            while (top != -1 && precedence(stack[top]) >= precedence(current)) {
                postfix[j++] = stack[top--];
            }
            stack[++top] = current;
        }
    }

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

    postfix[j] = '\0';  // Null terminate the postfix expression
    printf("%s\n", postfix);
}

int main() {
    // Read input infix expression
    char infix[MAX];
    fgets(infix, MAX, stdin);
    infix[strcspn(infix, "\n")] = 0;  // Remove newline character from input if present

    // Convert infix to postfix and output the result
    infixToPostfix(infix);
```

```
    return 0;
}
```

Status : Correct *Marks : 10/10*