Rajalakshmi Engineering College

Name: Pranav Shanmugam
Email: 240701395@rajalakshmi.edu.in
Roll no: 2116240701395
Phone: 8925357178
Branch: REC
Department: I CSE FD
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

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

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

typedef struct Stack {
   char data;
   struct Stack* next;
} Stack;

Stack* push(Stack* top, char value) {
   Stack* newNode = (Stack*)malloc(sizeof(Stack));
   newNode->data = value;
   newNode->next = top;
   return newNode;
}

Stack* pop(Stack* top) {
   if (!top) return NULL;
   Stack* temp = top;
   top = top->next;
   free(temp);
   return top;
}

char peek(Stack* top) {
   return top ? top->data : '\0';
}

int precedence(char op) {
   if (op == '+' || op == '-') return 1;
```

```
    if (op == '*' || op == '/') return 2;
    return 0;
}

void infixToPostfix(char* infix, char* postfix) {
    Stack* stack = NULL;
    int j = 0;

    for (int i = 0; i < strlen(infix); i++) {
        if (isdigit(infix[i])) postfix[j++] = infix[i];
        else if (infix[i] == '(') stack = push(stack, infix[i]);
        else if (infix[i] == ')') {
            while (peek(stack) != '(') {
                postfix[j++] = peek(stack);
                stack = pop(stack);
            }
            stack = pop(stack);
        } else {
            while (stack && precedence(peek(stack)) >= precedence(infix[i])) {
                postfix[j++] = peek(stack);
                stack = pop(stack);
            }
            stack = push(stack, infix[i]);
        }
    }

    while (stack) {
        postfix[j++] = peek(stack);
        stack = pop(stack);
    }

    postfix[j] = '\0';
}

int main() {
    char infix[31], postfix[31];
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    printf("%s\n", postfix);

    return 0;
}
```

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

2. 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

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

typedef struct Stack {
  char data;
```

```
    struct Stack* next;
} Stack;

Stack* push(Stack* top, char value) {
    Stack* newNode = (Stack*)malloc(sizeof(Stack));
    newNode->data = value;
    newNode->next = top;
    return newNode;
}

Stack* pop(Stack* top) {
    if (!top) return NULL;
    Stack* temp = top;
    top = top->next;
    free(temp);
    return top;
}

char peek(Stack* top) {
    return top ? top->data : '\0';
}

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

void infixToPostfix(char* infix, char* postfix) {
    Stack* stack = NULL;
    int j = 0;

    for (int i = 0; i < strlen(infix); i++) {
        if (isdigit(infix[i])) postfix[j++] = infix[i];
        else if (infix[i] == ' ') continue;
        else if (infix[i] == '(') stack = push(stack, infix[i]);
        else if (infix[i] == ')') {
            while (peek(stack) != '(') {
                postfix[j++] = peek(stack);
                stack = pop(stack);
            }
            stack = pop(stack);
        } else {
            while (stack && precedence(peek(stack)) >= precedence(infix[i])) {
                postfix[j++] = peek(stack);
                stack = pop(stack);
            }
            stack = push(stack, infix[i]);
        }
    }

    while (stack) {
        postfix[j++] = peek(stack);
        stack = pop(stack);
    }

    postfix[j] = '\0';
}

int main() {
    char infix[51], postfix[51];
    fgets(infix, sizeof(infix), stdin);

    infixToPostfix(infix, postfix);
    printf("%s\n", postfix);

    return 0;
}
```

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

3. 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

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

typedef struct Stack {
   char data;
   struct Stack* next;
} Stack;

Stack* push(Stack* top, char value) {
   Stack* newNode = (Stack*)malloc(sizeof(Stack));
   newNode->data = value;
   newNode->next = top;
   return newNode;
}
```

```
Stack* pop(Stack* top) {
    if (!top) return NULL;
    Stack* temp = top;
    top = top->next;
    free(temp);
    return top;
}

char peek(Stack* top) {
    return top ? top->data : '\0';
}

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

void infixToPostfix(char* infix, char* postfix) {
    Stack* stack = NULL;
    int j = 0;

    for (int i = 0; i < strlen(infix); i++) {
        if (isalnum(infix[i])) postfix[j++] = infix[i];
        else if (infix[i] == '(') stack = push(stack, infix[i]);
        else if (infix[i] == ')') {
            while (peek(stack) != '(') {
                postfix[j++] = peek(stack);
                stack = pop(stack);
            }
            stack = pop(stack);
        } else {
            while (stack && precedence(peek(stack)) >= precedence(infix[i])) {
                postfix[j++] = peek(stack);
                stack = pop(stack);
            }
            stack = push(stack, infix[i]);
        }
    }

    while (stack) {
        postfix[j++] = peek(stack);
        stack = pop(stack);
    }

    postfix[j] = '\0';
}

int main() {
    char infix[101], postfix[101];
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    printf("%s\n", postfix);

    return 0;
}
```

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