Rajalakshmi Engineering College

Name: Pavithra J
Email: 240701381@rajalakshmi.edu.in
Roll no: 240701381
Phone: 9363364978
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

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 <ctype.h>
#include <string.h>

#define MAX 100

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

// Push to stack
void push(char ch) {
    if (top < MAX - 1) {
        stack[++top] = ch;
    }
}

// Pop from stack
char pop() {
    if (top >= 0) {
        return stack[top--];
    }
    return '\0';
}

// Peek top of stack
char peek() {
    if (top >= 0) {
        return stack[top];
    }
    return '\0';
}

// Check if character is operator
int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

// Get precedence of operator
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 i = 0, k = 0;
    char ch;

    while ((ch = infix[i++]) != '\0') {
        if (ch == ' ') {
            continue; // Skip spaces
        } else if (isdigit(ch)) {
            postfix[k++] = ch;
        } else if (ch == '(') {
            push(ch);
        } else if (ch == ')') {
            while (peek() != '(' && top != -1) {
                postfix[k++] = pop();
            }
            pop(); // Discard '('
        } else if (isOperator(ch)) {
            while (top != -1 && precedence(peek()) >= precedence(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }

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

    postfix[k] = '\0'; // Null terminate the postfix expression

    printf("%s\n", postfix);
}

int main() {
    char infix[51];
    fgets(infix, sizeof(infix), stdin);  // Read input line with spaces
    infixToPostfix(infix);
    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;
};

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

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

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

// Peek operation
int peek(struct Node* top) {
   if (top != NULL) {
      return top->data;
   }
   return -1; // Stack is empty
}

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

   struct Node* top = NULL;

   // Push elements in order: a, b, c, d
   push(&top, a);
   push(&top, b);
   push(&top, c);
   push(&top, d);

   // Display stack after all pushes
   display(top);
   printf("\n");

   // Pop the top element (no output for this line)
   pop(&top);
   printf("\n");

   // Display stack after pop
   display(top);
```

```
    printf("\n");

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

    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

```
#include <stdio.h>
#include <ctype.h>  // For isdigit()
```

```
#include <string.h>

#define MAX 100

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

// Push to stack
void push(char ch) {
    if (top < MAX - 1) {
        stack[++top] = ch;
    }
}

// Pop from stack
char pop() {
    if (top >= 0) {
        return stack[top--];
    }
    return '\0';
}

// Peek top of stack
char peek() {
    if (top >= 0) {
        return stack[top];
    }
    return '\0';
}

// Check if character is operator
int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

// Get precedence of operator
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 i = 0, k = 0;
    char ch;

    while ((ch = infix[i++]) != '\0') {
        if (isdigit(ch)) {
            postfix[k++] = ch;
        }
        else if (ch == '(') {
            push(ch);
        }
        else if (ch == ')') {
            while (peek() != '(' && top != -1) {
                postfix[k++] = pop();
            }
            pop(); // Remove '('
        }
        else if (isOperator(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';

    // Output the postfix expression
    printf("%s\n", postfix);
}

// Main function to run the test
int main() {
    char infix[31];
    scanf("%s", infix);  // Read input expression
    infixToPostfix(infix);
    return 0;
}
```

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