Rajalakshmi Engineering College

Name: Jhanani shree
Email: 240701215@rajalakshmi.edu.in
Roll no: 240701215
Phone: 7373333511
Branch: REC
Department: I CSE AH
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 : 7.5

Section 1 : Coding

1. 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 <ctype.h>
#include <string.h>
typedef struct {
    char items[MAX];
    int top;
} Stack;
Stack* createStack() {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->top = -1;
    return stack;
}
int isEmpty(Stack* stack) {
    return stack->top == -1;
}
void push(Stack* stack, char item) {
    if (stack->top < MAX - 1) {
        stack->items[++stack->top] = item;
    }
```

```
}
char pop(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top--];
    }
    return '\0'; // Return null character if stack is empty
}
char peek(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top];
    }
    return '\0'; // Return null character if stack is empty
}
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return 0;
    }
}
void infixToPostfix(const char* expression, char* output) {
    Stack* stack = createStack();
    int j = 0;

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

        if (isalnum(charAtI)) { // If the character is an operand
            output[j++] = charAtI;
        } else if (charAtI == '(') { // If the character is '('
            push(stack, charAtI);
        } else if (charAtI == ')') { // If the character is ')'
            while (!isEmpty(stack) && peek(stack) != '(') {
                output[j++] = pop(stack);
            }
            pop(stack); // Pop '('
        } else { // The character is an operator
```

```
            while (!isEmpty(stack) && precedence(charAtI) <=
precedence(peek(stack))) {
                output[j++] = pop(stack);
            }
            push(stack, charAtI);
        }
    }
    while (!isEmpty(stack)) {
        output[j++] = pop(stack);
    }
    output[j] = '\0';
    free(stack);
}

int main() {
    char infix[MAX];
    char postfix[MAX];
    strcpy(infix, "(3+4)5");
    infixToPostfix(infix, postfix);
    printf("Input: %s\nOutput: %s\n\n", infix, postfix);
    strcpy(infix, "A+B*C-D/E");
    infixToPostfix(infix, postfix);
    printf("Input: %s\nOutput: %s\n\n", infix, postfix);
    strcpy(infix, "3+4*5/(6-2)");
    infixToPostfix(infix, postfix);
    printf("Input: %s\nOutput: %s\n\n", infix, postfix);

    return 0;
}
```

Status : Wrong ***Marks : 0/10***

2. Problem Statement

Raj is a software developer, and his team is building an application that processes user inputs in the form of strings containing brackets. One of the essential features of the application is to validate whether the input string meets specific criteria.

During testing, Raj inputs the string "(([])){}". The application correctly

returns "Valid string" because the input satisfies the criteria: every opening bracket (, [, and { has a corresponding closing bracket),], and }, arranged in the correct order.

Next, Raj tests the application with the string "([)]". This time, the application correctly returns "Invalid string" because the opening bracket [is incorrectly closed by the bracket), which violates the validation rules.

Finally, Raj enters the string "{[()]}". The application correctly identifies it as a "Valid string" since all opening brackets are matched with the corresponding closing brackets in the correct order.

As a software developer, Raj's responsibility is to ensure that the application works reliably and produces accurate results for all input strings, following the validation rules. He accomplishes this by using a method for solving such problems.

Input Format

The input comprises a string representing a sequence of brackets that need to be validated.

Output Format

The output prints "Valid string" if the string is valid. Otherwise, it prints "Invalid string".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: (([]))\{\}
Output: Valid string

Answer

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

```
#define MAX 100

// Stack structure
typedef struct {
    char items[MAX];
    int top;
} Stack;

// Function to create a stack
Stack* createStack() {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->top = -1;
    return stack;
}

// Function to check if the stack is empty
bool isEmpty(Stack* stack) {
    return stack->top == -1;
}

// Function to push an item onto the stack
void push(Stack* stack, char item) {
    if (stack->top < MAX - 1) {
        stack->items[++stack->top] = item;
    }
}

// Function to pop an item from the stack
char pop(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top--];
    }
    return '\0'; // Return null character if stack is empty
}

// Function to check if the brackets are balanced
bool isValid(const char* str) {
    Stack* stack = createStack();

    for (int i = 0; str[i] != '\0'; i++) {
        char charAtI = str[i];
```

```
        if (charAtI == '(' || charAtI == '[' || charAtI == '{') {
            push(stack, charAtI);
        } else if (charAtI == ')' || charAtI == ']' || charAtI == '}') {
            if (isEmpty(stack)) {
                return false; // No matching opening bracket
            }
            char top = pop(stack);
            if ((charAtI == ')' && top != '(') ||
                (charAtI == ']' && top != '[') ||
                (charAtI == '}' && top != '{')) {
                return false; // Mismatched brackets
            }
        }
    }

    bool balanced = isEmpty(stack); // If stack is empty, brackets are balanced
    free(stack); // Free the allocated stack memory
    return balanced;
}

int main() {
    char input[MAX];
    printf(" %s %s\n", input, isValid(input) ? "Valid string" : "Invalid string");


    return 0;
}
```

Status : Partially correct ***Marks : 7.5/10***

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 <stdlib.h>
#include <string.h>
#include <ctype.h>
typedef struct {
    char items[MAX];
    int top;
} Stack;
Stack* createStack() {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->top = -1;
    return stack;
}
int isEmpty(Stack* stack) {
    return stack->top == -1;
}
void push(Stack* stack, char item) {
    if (stack->top < MAX - 1) {
        stack->items[++stack->top] = item;
    }
}
char pop(Stack* stack) {
    if (!isEmpty(stack)) {
```

```
        return stack->items[stack->top--];
    }
    return '\0';
}
char peek(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top];
    }
    return '\0';
}
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return 0;
    }
}
void infixToPostfix(const char* expression, char* output) {
    Stack* stack = createStack();
    int j = 0;

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

        if (isdigit(charAtI)) {
            output[j++] = charAtI;
        } else if (charAtI == '(') {
            push(stack, charAtI);
        } else if (charAtI == ')') {
            while (!isEmpty(stack) && peek(stack) != '(') {
                output[j++] = pop(stack);
            }
            pop(stack);
        } else {
            while (!isEmpty(stack) && precedence(charAtI) <=
precedence(peek(stack))) {
                output[j++] = pop(stack);
```

```
            }
            push(stack, charAtI);
        }
    }
    while (!isEmpty(stack)) {
        output[j++] = pop(stack);
    }
    output[j] = '\0';
    free(stack);
}
int main() {
    char infix[MAX];
    char postfix[MAX];
    strcpy(infix, "1+2*3/4-5");
    infixToPostfix(infix, postfix);
    printf("Input: %s\nOutput: %s\n", infix, postfix);

    strcpy(infix, "5+6-4*8/2");
    infixToPostfix(infix, postfix);
    printf("Input: %s\nOutput: %s\n", infix, postfix);

    return 0;
}
```

Status : Wrong ***Marks : 0/10***