Rajalakshmi Engineering College

Name: Siddesh Kumar L
Email: 240701512@rajalakshmi.edu.in
Roll no: 240701512
Phone: null
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

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>

#define MAX 100

int isValid(char* s) {
    char stack[MAX];
    int top = -1;

    for (int i = 0; i < strlen(s); i++) {
        char ch = s[i];
```

```
        if (ch == '(' || ch == '[' || ch == '{') {
            stack[++top] = ch;
        }
        else if (ch == ')' || ch == ']' || ch == '}') {
            if (top == -1) {
                return 0;
            }

            char last = stack[top--];

            if ((ch == ')' && last != '(') ||
                (ch == ']' && last != '[') ||
                (ch == '}' && last != '{')) {
                return 0;
            }
        }
    }

    return top == -1;
}

int main() {
    char str[MAX];


    scanf("%s", str);

    if (isValid(str)) {
        printf("Valid string\n");
    } else {
        printf("Invalid string\n");
    }

    return 0;
}
```

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

2. Problem Statement

In an educational setting, Professor Smith tasks Computer Science students with designing an algorithm to evaluate postfix expressions efficiently, fostering problem-solving skills and understanding of stack-based computations.

The program prompts users to input a postfix expression, evaluates it, and displays the result, aiding students in honing their coding abilities.

Input Format

The input consists of the postfix mathematical expression.

The expression will contain real numbers and mathematical operators (+, -, *, /), without any space.

Output Format

The output prints the result of evaluating the given postfix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 82/

Output: 4

Answer

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

#define MAX 100

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

void push(double value) {
    if (top >= MAX - 1) {
        printf("Stack overflow\n");
```

```
        exit(1);
    }
    stack[++top] = value;
}

double pop() {
    if (top < 0) {
        printf("Stack underflow\n");
        exit(1);
    }
    return stack[top--];
}

double apply_operator(double a, double b, char op) {
    switch (op) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return a / b;
        default:
            printf("Invalid operator: %c\n", op);
            exit(1);
    }
}

double evaluate_postfix(const char *expr) {
    for (int i = 0; expr[i] != '\0'; i++) {
        char ch = expr[i];

        if (isdigit(ch)) {
            push(ch - '0'); // Single digit only
        } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            double b = pop();
            double a = pop();
            double result = apply_operator(a, b, ch);
            push(result);
        } else {
            printf("Invalid character in expression: %c\n", ch);
            exit(1);
        }
    }
```

```
    return pop();
}

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

    double result = evaluate_postfix(expr);

    if (result == (int)result)
        printf("%d\n", (int)result);
    else
        printf("%lf\n", result);

    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>

#define MAX 100

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

void push(char ch) {
    stack[++top] = ch;
}

char pop() {
    return (top == -1) ? '\0' : stack[top--];
}

char peek() {
    return (top == -1) ? '\0' : stack[top];
}

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

```
}

int is_operator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

void infix_to_postfix(const char* infix, char* postfix) {
    int i = 0, j = 0;
    char ch;

    while ((ch = infix[i++]) != '\0') {
        if (isalnum(ch)) {
            postfix[j++] = ch;
        }
        else if (ch == '(') {
            push(ch);
        }
        else if (ch == ')') {
            while (peek() != '(') {
                postfix[j++] = pop();
            }
            pop(); // remove '('
        }
        else if (is_operator(ch)) {
            while (precedence(peek()) >= precedence(ch)) {
                postfix[j++] = pop();
            }
            push(ch);
        }
    }

    while (top != -1) {
        postfix[j++] = pop();
    }

    postfix[j] = '\0';
}

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

```
    infix_to_postfix(infix, postfix);

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

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