Rajalakshmi Engineering College

Name: sugitha nesamani
Email: 241801279@rajalakshmi.edu.in
Roll no: 241801279
Phone: 8637611457
Branch: REC
Department: I AI & DS AF
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

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
struct Stack {
    char arr[MAX];
    int top;
};
void init(struct Stack* s) {
    s->top = -1;
}
```

```
void push(struct Stack* s, char ch) {
    if (s->top < MAX - 1)
        s->arr[++(s->top)] = ch;
}


char pop(struct Stack* s) {
    if (s->top == -1)
        return '\0';
    return s->arr[(s->top)--];
}


char peek(struct Stack* s) {
    if (s->top == -1)
        return '\0';
    return s->arr[s->top];
}


int isMatchingPair(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '[' && close == ']') ||
           (open == '{' && close == '}');
}


int isValid(char* str) {
    struct Stack stack;
    init(&stack);
    int i;
    for (i = 0; str[i]; i++) {
        if (str[i] == '(' || str[i] == '[' || str[i] == '{') {
            push(&stack, str[i]);
        } else if (str[i] == ')' || str[i] == ']' || str[i] == '}') {
            char top = pop(&stack);
            if (!isMatchingPair(top, str[i])) {
                return 0; // Mismatch
            }
        }
```

```
    }
    return stack.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

Rithi is building a simple text editor that allows users to type characters, undo their typing, and view the current text. She has implemented this text editor using an array-based stack data structure.

She has to develop a basic text editor with the following features:

Type a Character (Push): Users can type a character and add it to the text editor.Undo Typing (Pop): Users can undo their typing by removing the last character they entered from the editor.View Current Text (Display): Users can view the current text in the editor, which is the sequence of characters in the buffer.Exit: Users can exit the text editor application.

Write a program that simulates this text editor's undo feature using a character stack and implements the push, pop and display operations accordingly.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the character to be pushed onto the stack.

Choice 2: Pop the character from the stack.

Choice 3: Display the characters in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

1. If the choice is 1, print: "Typed character: <character>" where <character> is the character that was pushed to the stack.
2. If the choice is 2, print: "Undo: Removed character <character>" where <character> is the character that was removed from the stack.
3. If the choice is 2, and if the stack is empty without any characters, print "Text editor buffer is empty. Nothing to undo."
4. If the choice is 3, print: "Current text: <character1> <character2> ... <characterN>" where <character1>, <character2>, ... are the characters in the stack, starting from the last pushed character.
5. If the choice is 3, and there are no characters in the stack, print "Text editor buffer is empty."
6. If the choice is 4, exit the program.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1 H
1 A
3
4
Output: Typed character: H

Typed character: A
Current text: A H

Answer

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

#define MAX 100

char stack[MAX];
int top = -1;
void push(char ch) {
    if (top < MAX - 1) {
        stack[++top] = ch;
        printf("Typed character: %c\n", ch);
    }
}
void pop() {
    if (top == -1) {
        printf("Text editor buffer is empty. Nothing to undo.\n");
    } else {
        printf("Undo: Removed character %c\n", stack[top--]);
    }
}
void display() {
    if (top == -1) {
        printf("Text editor buffer is empty.\n");
    } else {
        printf("Current text: ");
        for (int i = top; i >= 0; i--) {
            printf("%c ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice;
    char ch;

    while (1) {
```

```
        if (scanf("%d", &choice) != 1) break;

        switch (choice) {
            case 1:
                scanf(" %c", &ch);
                push(ch);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
                break;
        }
    }

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

#define MAX 100
char stack[MAX];
int top = -1;
void push(char op) {
    stack[++top] = op;
}

char pop() {
    return stack[top--];
}

char peek() {
    return stack[top];
}


int isEmpty() {
    return top == -1;
}

int precedence(char op) {
```

```
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}

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


void removeSpaces(char* str) {
    int i = 0, j = 0;
    while (str[i]) {
        if (str[i] != ' ')
            str[j++] = str[i];
        i++;
    }
    str[j] = '\0';
}

void infixToPostfix(char* infix, char* postfix) {
    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 (!isEmpty() && peek() != '(')
                postfix[k++] = pop();
            if (!isEmpty() && peek() == '(')
                pop(); // Remove '('
        } else if (isOperator(ch)) {
            while (!isEmpty() && precedence(peek()) >= precedence(ch))
                postfix[k++] = pop();
            push(ch);
        }
    }
```

```
    while (!isEmpty())
        postfix[k++] = pop();

    postfix[k] = '\0';
}

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

    fgets(infix, sizeof(infix), stdin);
    removeSpaces(infix);

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

    return 0;
}
```

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