Rajalakshmi Engineering College

Name: Santhosh G
Email: 240701473@rajalakshmi.edu.in
Roll no: 240701473
Phone: 8883772237
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

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<string.h>
#define MAX 100
int stack[MAX],top= -1;
char expr[MAX], post[MAX];
void push (char sym);
char pop();
char Top();
int priority (char sym);
int main(){
   fgets(expr, MAX, stdin);
   for(int i=0;i<strlen(expr);i++){
      if( expr[i]==' '){
         continue;
      }
      if( expr[i]>='0' && expr[i]<='9'){
         printf("%c",expr[i]);
      }
      else if(expr[i]=='('){
         push(expr[i]);
      }
      else if(expr[i]==')'){
         while(Top()!='('){
            printf("%c",pop());
         }
         pop();
      }
      else{
         while(priority(expr[i])<=priority(Top()) && top!=-1){
                printf("%c",pop());
            }
            push(expr[i]);
        }
    }
    for(int i=top;i>=0;i--){
        printf("%c",pop());
    }
    return 0;
}

void push(char sym){
    top+=1;
    stack[top]=sym;
}

char pop(){
    char e;
    e=stack[top];
    top-=1;
    return e;
}

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

int priority(char sym){
    int p=0;
    switch(sym){
        case '(':
            p=-1;
            break;
        case '+':
        case '-':
            p=1;
            break;
        case '*':
        case '/':
        case '%':
            p=2;
            break;
        case '^':
            p=3;
            break;
    }
    return p;
}
```

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

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: (([])){&#125;
Output: Valid string

Answer

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

#define MAX 100


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


void initStack(Stack *s) {
   s->top = -1;
}


int isEmpty(Stack *s) {
   return s->top == -1;
}


void push(Stack *s, char c) {
   if (s->top == MAX - 1) {
      printf("Stack overflow\n");
      return;
   }
```

```
    s->items[++(s->top)] = c;
}


char pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack underflow\n");
        return '\0';
    }
    return s->items[(s->top)--];
}


int isBalanced(char *expr) {
    Stack stack;
    initStack(&stack);

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


        if (ch == '(' || ch == '{' || ch == '[') {
            push(&stack, ch);
        } else if (ch == ')' || ch == '}' || ch == ']') {
            if (isEmpty(&stack)) return 0;

            char top = pop(&stack);


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

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

    if (isBalanced(expr)) {
        printf("Valid string\n");
    } else {
        printf("Invalid string\n");
    }

    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

```
// You are using GCC
// You are using GCC
#include<stdio.h>
#include<string.h>
#define MAX 100
int stack[MAX],top= -1;
char expr[MAX], post[MAX];
void push (char sym);
char pop();
char Top();
int priority (char sym);
int main(){
   fgets(expr, MAX, stdin);
   for(int i=0;i<strlen(expr);i++){
      if ( expr[i]==' '){
         continue;
      }
      if( expr[i]>='0' && expr[i]<='9' || (expr[i]>='a' && expr[i]<='z')||(expr[i]>='A' &&
expr[i]<='Z')){
         printf("%c",expr[i]);
      }
      else if(expr[i]=='('){
         push(expr[i]);
      }
      else if(expr[i]==')'){
         while(Top()!='('){
            printf("%c",pop());
         }
         pop();
      }
      else{
            while(priority(expr[i])<=priority(Top()) && top!=-1){
                printf("%c",pop());
            }
            push(expr[i]);
        }
    }
    for(int i=top;i>=0;i--){
        printf("%c",pop());
    }
    return 0;
}

void push(char sym){
    top+=1;
    stack[top]=sym;
}

char pop(){
    char e;
    e=stack[top];
    top-=1;
    return e;
}

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

int priority(char sym){
    int p=0;
    switch(sym){
        case '(':
            p=-1;
            break;
        case '+':
        case '-':
            p=1;
            break;
        case '*':
        case '/':
        case '%':
            p=2;
            break;
        case '^':
            p=3;
            break;
    }
    return p;
}
```

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