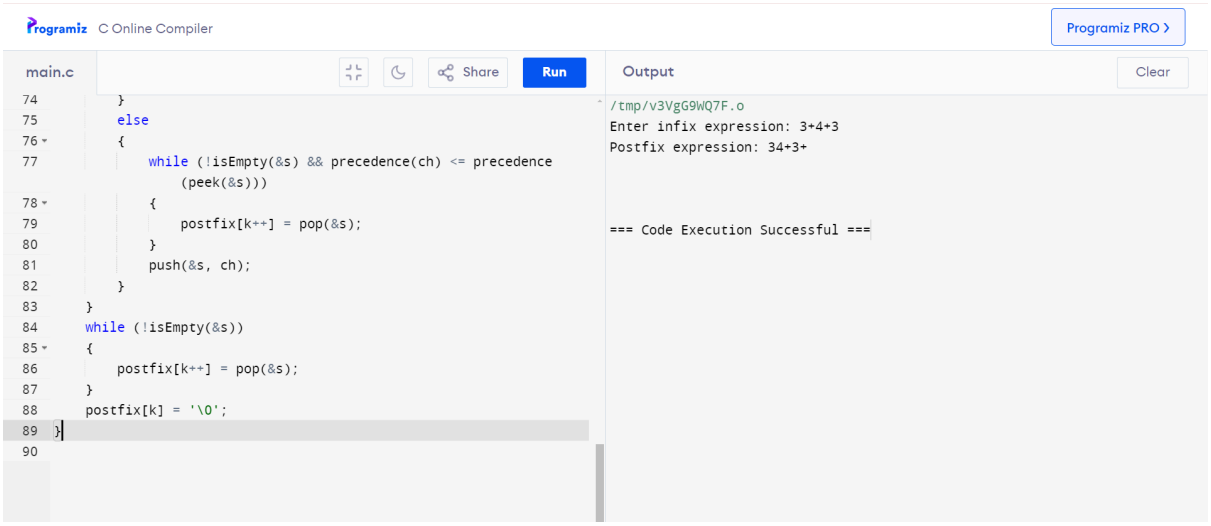
//Implement stack for infix to postfix expression

```
#include <stdio.h>
#define MAX 100
typedef struct
{
    int top;
    char items[MAX];
}Stack;
void init(Stack *s);
int isEmpty(Stack *s);
void push(Stack *s, char item);
char pop(Stack *s);
char peek(Stack *s);
int precedence(char op);
void infixToPostfix(char *infix, char *postfix);
int main()
{
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    fgets(infix, MAX, stdin);
    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
void init(Stack *s)
{
    s->top = -1;
}
int isEmpty(Stack *s)
{
    return s->top == -1;
}
void push(Stack *s, char item)
{
    s->items[++(s->top)] = item;
}
char pop(Stack *s)
{
    return s->items[(s->top)--];
}
char peek(Stack *s)
{
    return s->items[s->top];
}
int precedence(char op)
{
```

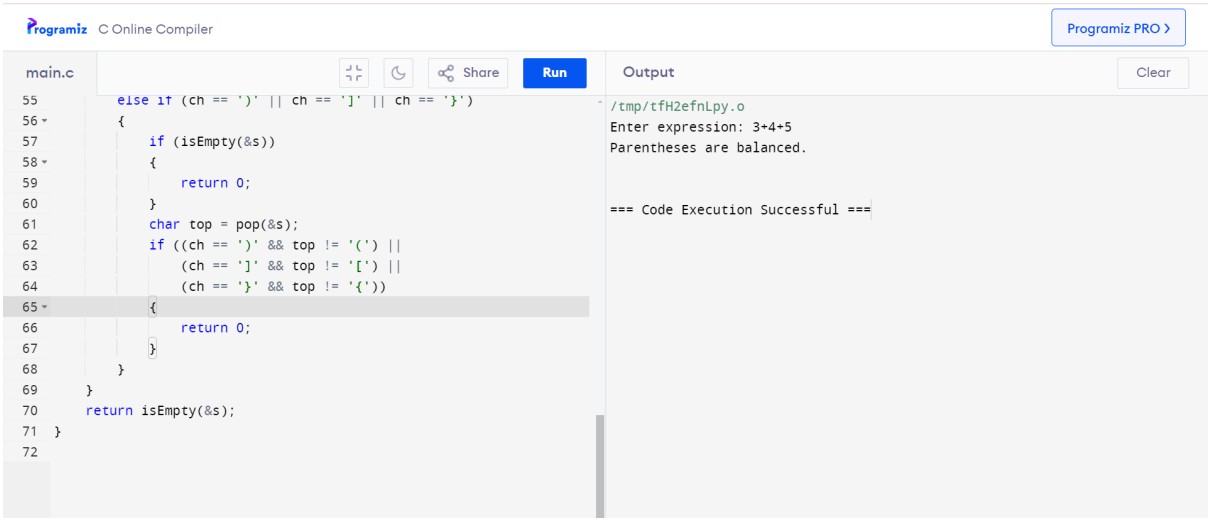
```
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return 0;
}
void infixToPostfix(char *infix, char *postfix)
{
    Stack s;
    init(&s);
    int k = 0;
    for (int i = 0; infix[i] != '\0'; i++)
    {
        char ch = infix[i];
        if (ch >= '0' && ch <= '9')
        {
            postfix[k++] = ch;
        }
        else if (ch == '(')
        {
            push(&s, ch);
        }
        else if (ch == ')')
        {
            while (!isEmpty(&s) && peek(&s) != '(')
            {
                postfix[k++] = pop(&s);
            }
            pop(&s);
        }
        else
        {
            while (!isEmpty(&s) && precedence(ch) <= precedence(peek(&s)))
            {
                postfix[k++] = pop(&s);
            }
            push(&s, ch);
        }
    }
    while (!isEmpty(&s))
    {
        postfix[k++] = pop(&s);
    }
    postfix[k] = '\0';
}
```



//Balancing the parenthesis

```
#include <stdio.h>
#define MAX 100
typedef struct
{
    int top;
    char items[MAX];
}Stack;
void init(Stack *s);
int isEmpty(Stack *s);
void push(Stack *s, char item);
char pop(Stack *s);
int isBalanced(char *expression);
int main()
{
    char expression[MAX];
    printf("Enter expression: ");
    fgets(expression, MAX, stdin);
    if (isBalanced(expression))
    {
        printf("Parentheses are balanced.\n");
    }
    else
    {
        printf("Parentheses are not balanced.\n");
    }
    return 0;
}
void init(Stack *s)
{
    s->top = -1;
}
```

```
int isEmpty(Stack *s)
{
    return s->top == -1;
}
void push(Stack *s, char item)
{
    s->items[++(s->top)] = item;
}
char pop(Stack *s)
{
    return s->items[(s->top)--];
}
int isBalanced(char *expression)
{
    Stack s;
    init(&s);
    for (int i = 0; expression[i] != '\0'; i++)
    {
        char ch = expression[i];
        if (ch == '(' || ch == '[' || ch == '{')
        {
            push(&s, ch);
        }
        else if (ch == ')' || ch == ']' || ch == '}')
        {
            if (isEmpty(&s))
            {
                return 0;
            }
            char top = pop(&s);
            if ((ch == ')' && top != '(') ||
                (ch == ']' && top != '[') ||
                (ch == '}' && top != '{'))
            {
                return 0;
            }
        }
    }
    return isEmpty(&s);
}
```


Programiz C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear
else if (ch == ')' || ch == ']' || ch == '}')
{
if (isEmpty(&s))
{
return 0;
}
char top = pop(&s);
if ((ch == ')' && top != '(') ||
(ch == ']' && top != '[') ||
(ch == '}' && top != '{'))
{
return 0;
}
}
}
return isEmpty(&s);
}
/tmp/tfH2efnLpy.o
Enter expression: 3+4+5
Parentheses are balanced.
=== Code Execution Successful ===