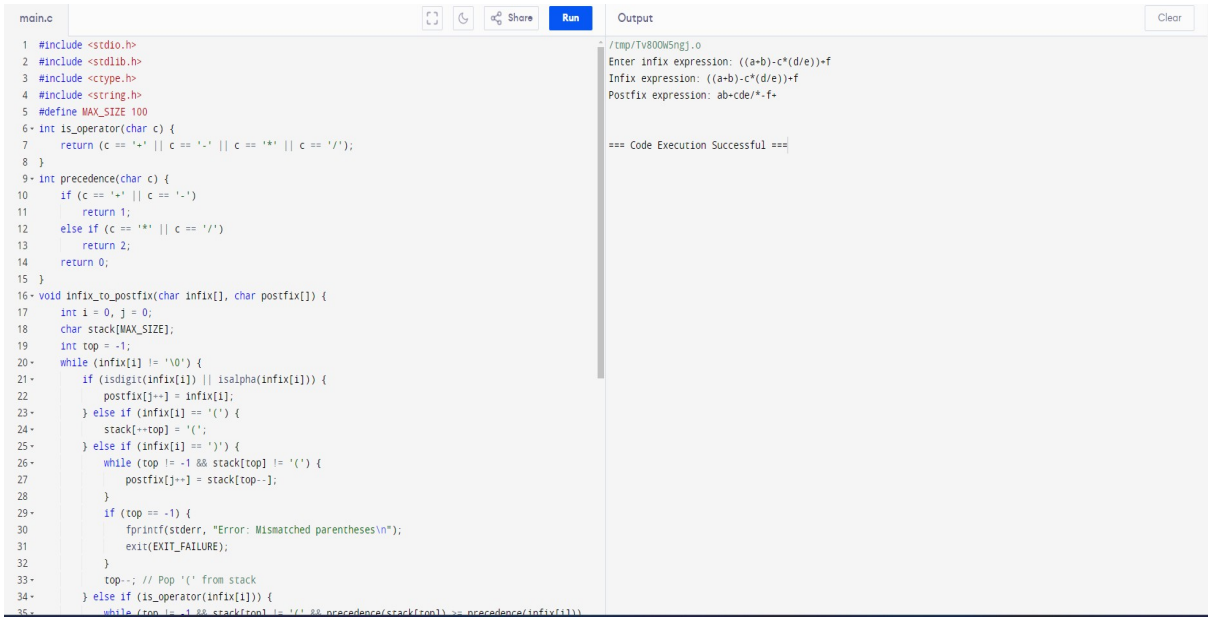
1.Write a c program for converting infix to postfix.



main.c

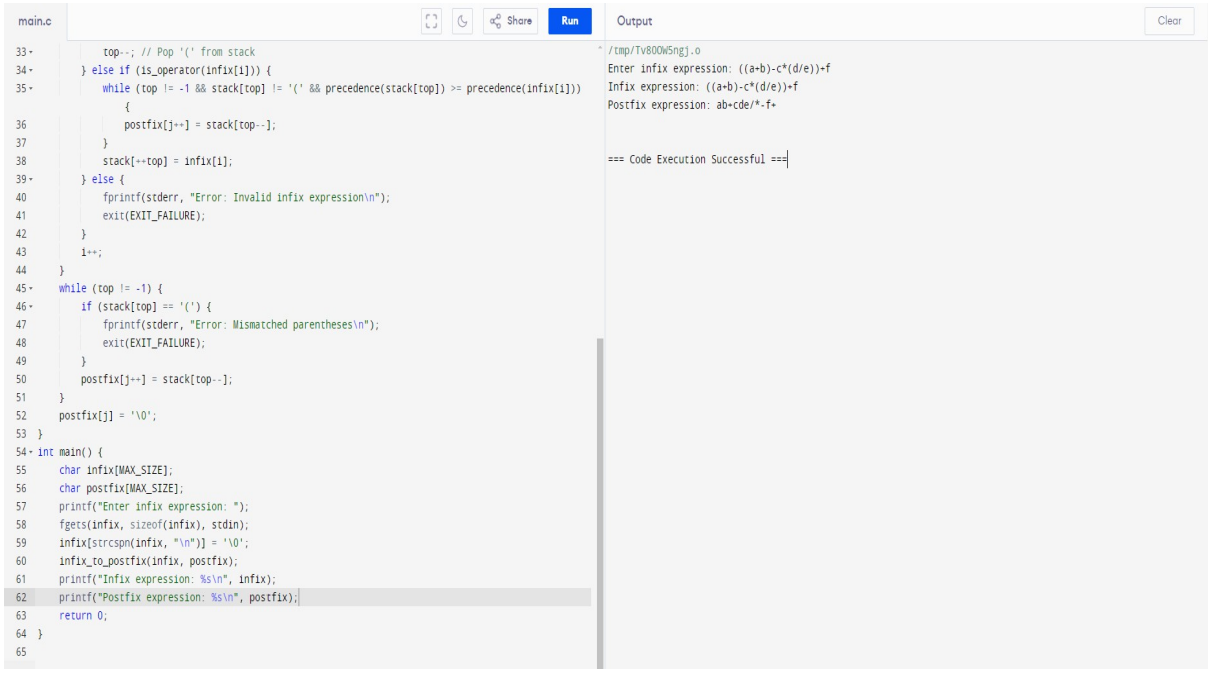
Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX_SIZE 100
int is_operator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}
int precedence(char c) {
    if (c == '+' || c == '-')
        return 1;
    else if (c == '*' || c == '/')
        return 2;
    return 0;
}
void infix_to_postfix(char infix[], char postfix[]) {
    int i = 0, j = 0;
    char stack[MAX_SIZE];
    int top = -1;
    while (infix[i] != '\0') {
        if (isdigit(infix[i]) || isalpha(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            stack[++top] = '(';
        } else if (infix[i] == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = stack[top--];
            }
            if (top == -1) {
                fprintf(stderr, "Error: Mismatched parentheses\n");
                exit(EXIT_FAILURE);
            }
            top--; // Pop '(' from stack
        } else if (is_operator(infix[i])) {
```

Output Clear

```
/tmp/Tv8OOW5ngj.o
Enter infix expression: ((a+b)-c*(d/e))+f
Infix expression: ((a+b)-c*(d/e))+f
Postfix expression: ab+cde/*-f+


=== Code Execution Successful ===
```



main.c

Share Run

```c
            top--; // Pop '(' from stack
        } else if (is_operator(infix[i])) {
            while (top != -1 && stack[top] != '(' && precedence(stack[top]) >= precedence(infix[i]))
                {
                postfix[j++] = stack[top--];
            }
            stack[++top] = infix[i];
        } else {
            fprintf(stderr, "Error: Invalid infix expression\n");
            exit(EXIT_FAILURE);
        }
        i++;
    }
    while (top != -1) {
        if (stack[top] == '(') {
            fprintf(stderr, "Error: Mismatched parentheses\n");
            exit(EXIT_FAILURE);
        }
        postfix[j++] = stack[top--];
    }
    postfix[j] = '\0';
}
int main() {
    char infix[MAX_SIZE];
    char postfix[MAX_SIZE];
    printf("Enter infix expression: ");
    fgets(infix, sizeof(infix), stdin);
    infix[strcspn(infix, "\n")] = '\0';
    infix_to_postfix(infix, postfix);
    printf("Infix expression: %s\n", infix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

Output Clear

```
/tmp/Tv8OOW5ngj.o
Enter infix expression: ((a+b)-c*(d/e))+f
Infix expression: ((a+b)-c*(d/e))+f
Postfix expression: ab+cde/*-f+


=== Code Execution Successful ===
```

2.Write a c program for converting postfix to infix.

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX_SIZE 100
struct Stack {
    char data[MAX_SIZE];
    int top;
};
void init_stack(struct Stack *s) {
    s->top = -1;
}
void push(struct Stack *s, char c) {
    if (s->top == MAX_SIZE - 1) {
        fprintf(stderr, "Error: Stack overflow\n");
        exit(EXIT_FAILURE);
    }
    s->data[++(s->top)] = c;
}
char pop(struct Stack *s) {
    if (s->top == -1) {
        fprintf(stderr, "Error: Stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return s->data[(s->top)--];
}
int is_operand(char c) {
    return isalnum(c);
}
void postfix_to_infix(char postfix[], char infix[]) {
    struct Stack stack;
    init_stack(&stack);
    int i = 0;
    while (postfix[i] != '\0') {
```

Output

```
/tmp/mEaFjujfbB.o
Enter postfix expression: ab+c*
Postfix expression: ab+c*
Infix expression: @(a+b)()*c))-


=== Code Execution Successful ===
```

main.c

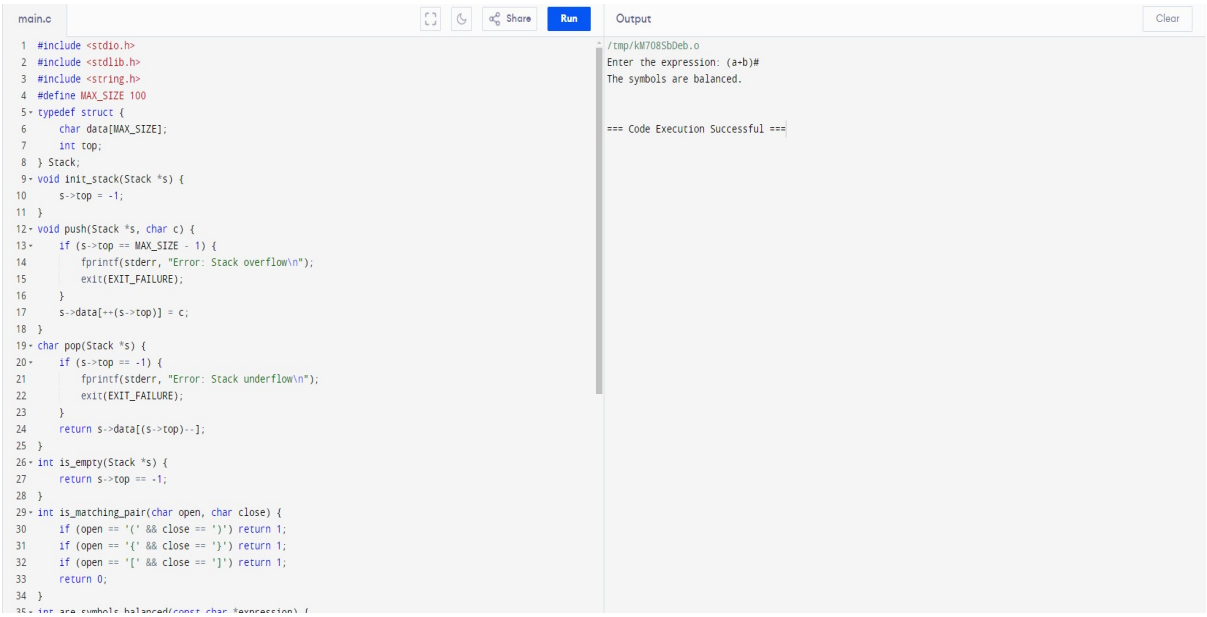
```c
}
void postfix_to_infix(char postfix[], char infix[]) {
    struct Stack stack;
    init_stack(&stack);
    int i = 0;
    while (postfix[i] != '\0') {
        if (is_operand(postfix[i])) {
            push(&stack, postfix[i]);
        } else if (postfix[i] == '+' || postfix[i] == '-' || postfix[i] == '*' || postfix[i] == '/')
            {
            char op1 = pop(&stack);
            char op2 = pop(&stack);
            sprintf(infix + strlen(infix), "(%c%c%c)", op2, postfix[i], op1);
            push(&stack, infix[strlen(infix) - 1]);
        }
        i++;
    }
    while (stack.top != -1) {
        infix[strlen(infix)] = pop(&stack);
    }
    infix[strlen(infix)] = '\0';
}
int main() {
    char postfix[MAX_SIZE];
    char infix[MAX_SIZE];
    printf("Enter postfix expression: ");
    fgets(postfix, sizeof(postfix), stdin);
    postfix[strcspn(postfix, "\n")] = '\0';
    postfix_to_infix(postfix, infix);
    printf("Postfix expression: %s\n", postfix);
    printf("Infix expression: %s\n", infix);
    return 0;
}
```

Output

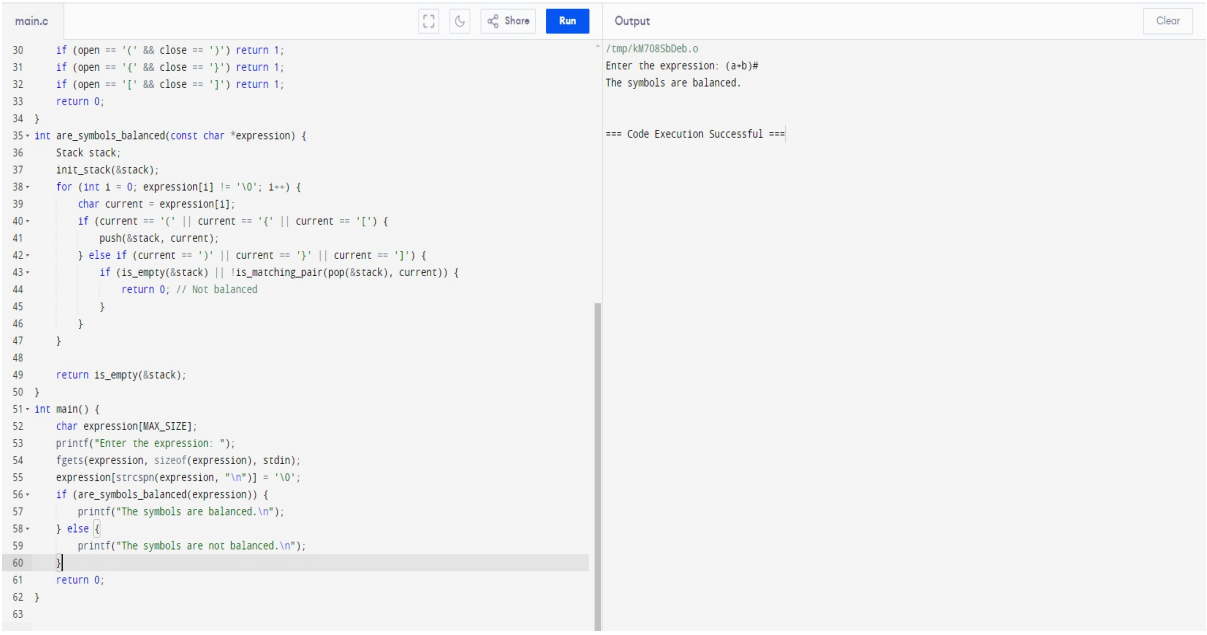
```
/tmp/mEaFjujfbB.o
Enter postfix expression: ab+c*
Postfix expression: ab+c*
Infix expression: @(a+b)()*c))-


=== Code Execution Successful ===
```

3.Write a c program for balancing Symbols.




```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SIZE 100
typedef struct {
    char data[MAX_SIZE];
    int top;
} Stack;
void init_stack(Stack *s) {
    s->top = -1;
}
void push(Stack *s, char c) {
    if (s->top == MAX_SIZE - 1) {
        fprintf(stderr, "Error: Stack overflow\n");
        exit(EXIT_FAILURE);
    }
    s->data[++(s->top)] = c;
}
char pop(Stack *s) {
    if (s->top == -1) {
        fprintf(stderr, "Error: Stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return s->data[(s->top)--];
}
int is_empty(Stack *s) {
    return s->top == -1;
}
int is_matching_pair(char open, char close) {
    if (open == '(' && close == ')') return 1;
    if (open == '{' && close == '}') return 1;
    if (open == '[' && close == ']') return 1;
    return 0;
}
```




```c
    if (open == '(' && close == ')') return 1;
    if (open == '{' && close == '}') return 1;
    if (open == '[' && close == ']') return 1;
    return 0;
}
int are_symbols_balanced(const char *expression) {
    Stack stack;
    init_stack(&stack);
    for (int i = 0; expression[i] != '\0'; i++) {
        char current = expression[i];
        if (current == '(' || current == '{' || current == '[') {
            push(&stack, current);
        } else if (current == ')' || current == '}' || current == ']') {
            if (is_empty(&stack) || !is_matching_pair(pop(&stack), current)) {
                return 0; // Not balanced
            }
        }
    }

    return is_empty(&stack);
}
int main() {
    char expression[MAX_SIZE];
    printf("Enter the expression: ");
    fgets(expression, sizeof(expression), stdin);
    expression[strcspn(expression, "\n")] = '\0';
    if (are_symbols_balanced(expression)) {
        printf("The symbols are balanced.\n");
    } else {
        printf("The symbols are not balanced.\n");
    }
    return 0;
}
```