```c
#include <stdio.h>
typedef struct {
    char arr[MAX][MAX];
    int top;
} Stack;
void initStack(Stack *s) {
    s->top = -1;
}
int isEmpty(Stack *s) {
    return s->top == -1;
}
void push(Stack *s, char *str) {
    if (s->top < MAX - 1) {
        s->top++;
        strcpy(s->arr[s->top], str);
    } else {
        printf("Stack overflow\n");
    }
}
void pop(Stack *s, char *str) {
    if (!isEmpty(s)) {
        strcpy(str, s->arr[s->top]);
        s->top--;
    } else {
        printf("Stack underflow\n");
    }
}
int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}
void postfixToInfix(char *postfix, char *infix) {
    Stack s;
    initStack(&s);
    char temp[MAX];
    int i, j;
    for (i = 0; postfix[i] != '\0'; i++) {
```

```c
    for (i = 0; postfix[i] != '\0'; i++) {
        if (isalnum(postfix[i])) {
            temp[0] = postfix[i];
            temp[1] = '\0';
            push(&s, temp);
        } else if (isOperator(postfix[i])) {
            char op2[MAX], op1[MAX];
            char result[MAX];
            pop(&s, op2);
            pop(&s, op1);
            snprintf(result, sizeof(result), "(%s%c%s)", op1, postfix[i], op2);
            push(&s, result);
        }
    }
    pop(&s, infix);
}
int main() {
    char postfix[MAX];
    char infix[MAX];

    printf("Enter a postfix expression: ");
    scanf("%s", postfix);

    postfixToInfix(postfix, infix);

    printf("Infix expression: %s\n", infix);

    return 0;
}
```

```c
#include <stdio.h>
typedef struct {
    char arr[MAX];
    int top;
} Stack;
void initStack(Stack *s) {
    s->top = -1;
}
int isEmpty(Stack *s) {
    return s->top == -1;
}
int isFull(Stack *s) {
    return s->top == MAX - 1;
}
void push(Stack *s, char ch) {
    if (!isFull(s)) {
        s->arr[++(s->top)] = ch;
    } else {
        printf("Stack overflow\n");
    }
}
char pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->arr[(s->top)--];
    } else {
        printf("Stack underflow\n");
        return '\0';
    }
}
char peek(Stack *s) {
    if (!isEmpty(s)) {
        return s->arr[s->top];
    } else {
        return '\0';
    }
}
```

```c
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}
int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^';
}
void infixToPostfix(char *infix, char *postfix) {
    Stack s;
    initStack(&s);
    int i = 0, j = 0;
    while (infix[i] != '\0') {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(&s, infix[i]);
        } else if (infix[i] == ')') {
            while (!isEmpty(&s) && peek(&s) != '(') {
                postfix[j++] = pop(&s);
            }
            if (!isEmpty(&s) && peek(&s) == '(') {
                pop(&s);
            }
        } else if (isOperator(infix[i])) {
            while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(infix[i])) {
                postfix[j++] = pop(&s);
```

```
            }
            push(&s, infix[i]);
        }
        i++;
    }
    while (!isEmpty(&s)) {
        postfix[j++] = pop(&s);
    }
    postfix[j] = '\0';
}
int main() {
    char infix[MAX];
    char postfix[MAX];

    printf("Enter an infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

```c
#include <stdio.h>
typedef struct {
    char arr[MAX];
    int top;
} Stack;
void initStack(Stack *s) {
    s->top = -1;
}
int isEmpty(Stack *s) {
    return s->top == -1;
}
int isFull(Stack *s) {
    return s->top == MAX - 1;
}
void push(Stack *s, char ch) {
    if (!isFull(s)) {
        s->arr[++(s->top)] = ch;
    } else {
        printf("Stack overflow\n");
    }
}
char pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->arr[(s->top)--];
    } else {
        printf("Stack underflow\n");
        return '\0';
    }
}
char peek(Stack *s) {
    if (!isEmpty(s)) {
        return s->arr[s->top];
    } else {
        return '\0';
    }
}
```

```c
int isOpeningSymbol(char ch) {
    return ch == '(' || ch == '{' || ch == '[';
}
int isClosingSymbol(char ch) {
    return ch == ')' || ch == '}' || ch == ']';
}
char matchingOpeningSymbol(char ch) {
    switch (ch) {
        case ')': return '(';
        case '}': return '{';
        case ']': return '[';
        default: return '\0';
    }
}
int areSymbolsBalanced(char *str) {
    Stack s;
    initStack(&s);
    int i;
    for (i = 0; str[i] != '\0'; i++) {
        char current = str[i];

        if (isOpeningSymbol(current)) {
            push(&s, current);
        } else if (isClosingSymbol(current)) {
            if (isEmpty(&s)) {
                return 0;
            }
            char top = pop(&s);
            if (top != matchingOpeningSymbol(current)) {
                return 0;
            }
        }
    }
```

```
    f
    return isEmpty(&s);
}
int main() {
    char str[MAX];
    printf("Enter a string with symbols: ");
    scanf("%s", str);

    if (areSymbolsBalanced(str)) {
        printf("The symbols are balanced.\n");
    } else {
        printf("The symbols are not balanced.\n");
    }
    return 0;
}
```