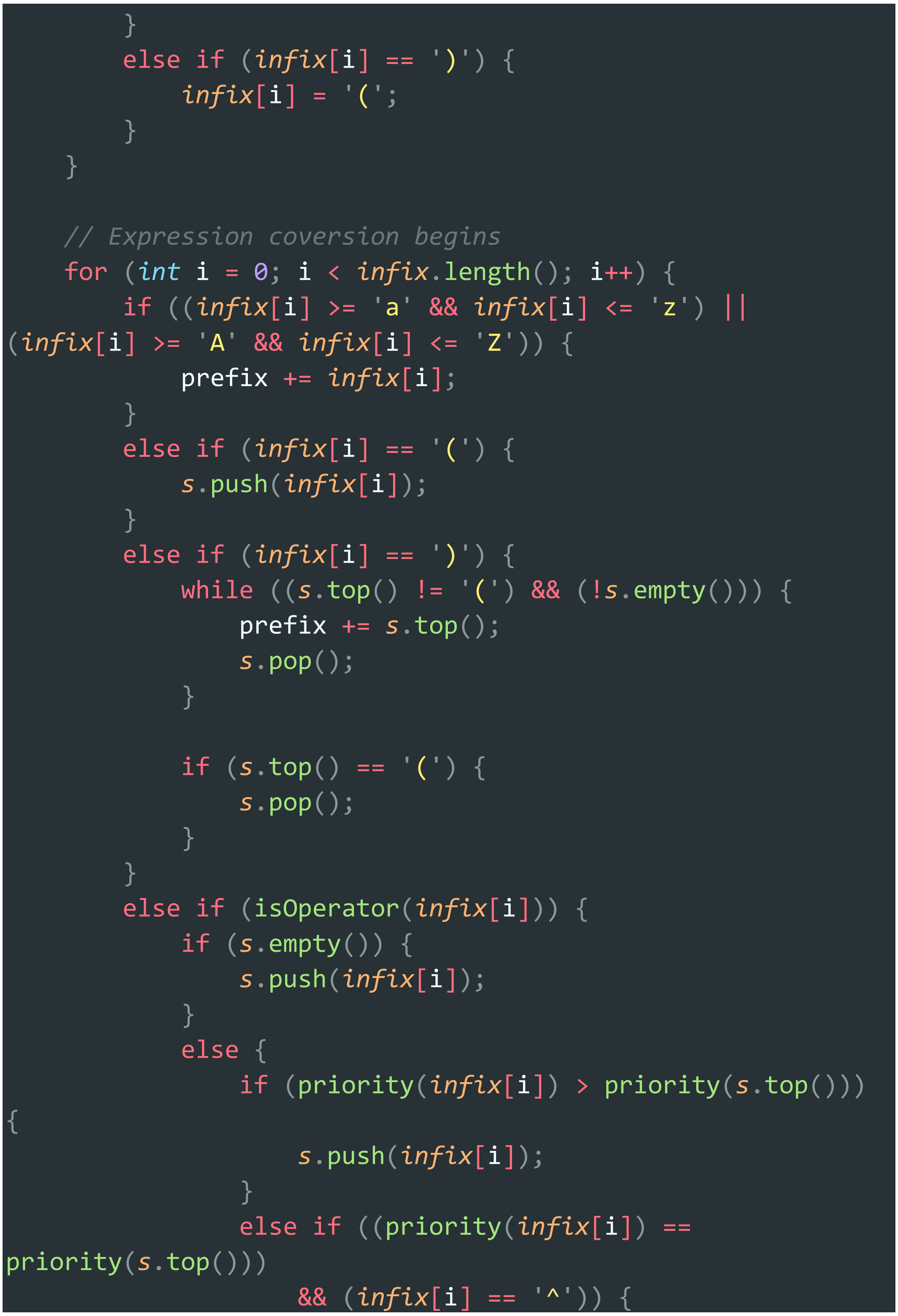
# INFIX TO PREFIX

```cpp
#include <iostream>
#include <stack>
#include <algorithm>

using namespace std;

bool isOperator(char c)
{
    if (c == '+' || c == '-' || c == '*' || c == '/' || c ==
'^') {
        return true;
    }
    else {
        return false;
    }
}

int priority(char c)
{
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}

string InfixToPrefix(stack<char> s, string infix)
{
    string prefix;
    reverse(infix.begin(), infix.end());

    //exchanging '(' with ')' and vice-versa
    for (int i = 0; i < infix.length(); i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
```

# INFIX TO PREFIX



```
        }
        else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }

    // Expression coversion begins
    for (int i = 0; i < infix.length(); i++) {
        if ((infix[i] >= 'a' && infix[i] <= 'z') ||
(infix[i] >= 'A' && infix[i] <= 'Z')) {
            prefix += infix[i];
        }
        else if (infix[i] == '(') {
            s.push(infix[i]);
        }
        else if (infix[i] == ')') {
            while ((s.top() != '(') && (!s.empty())) {
                prefix += s.top();
                s.pop();
            }

            if (s.top() == '(') {
                s.pop();
            }
        }
        else if (isOperator(infix[i])) {
            if (s.empty()) {
                s.push(infix[i]);
            }
            else {
                if (priority(infix[i]) > priority(s.top()))
{
                    s.push(infix[i]);
                }
                else if ((priority(infix[i]) ==
priority(s.top()))
                    && (infix[i] == '^')) {
```

# INFIX TO PREFIX

```
                    while ((priority(infix[i]) ==
priority(s.top()))
                        && (infix[i] == '^')) {
                        prefix += s.top();
                        s.pop();
                    }
                    s.push(infix[i]);
                }
                else if (priority(infix[i]) ==
priority(s.top())) {
                    s.push(infix[i]);
                }
                else {
                    while ((!s.empty()) &&
(priority(infix[i]) < priority(s.top()))) {
                        prefix += s.top();
                        s.pop();
                    }
                    s.push(infix[i]);
                }
            }
        }
    }

    while (!s.empty()) {
        prefix += s.top();
        s.pop();
    }

    reverse(prefix.begin(), prefix.end());
    return prefix;
}

int main()
{

    string infix, prefix;
    cout << "Enter a Infix Expression :" << endl;
```

## INFIX TO PREFIX

```cpp
    cin >> infix;
    stack<char> stack;
    cout << "INFIX EXPRESSION: " << infix << endl;
    prefix = InfixToPrefix(stack, infix);
    cout << endl
         << "PREFIX EXPRESSION: " << prefix;

    return 0;
}
```