```
#include <iostream>

#include <stack>

#include <string>

using namespace std;




struct Node {

char data;

Node* next;

};




class Stack {

private:

Node* top;

public:
```

```
Stack() : top(nullptr) {}



void push(char value) {

Node* newNode = new Node{value, top};

top = newNode;

}



char pop() {

if (!isEmpty()) {

char value = top->data;

Node* temp = top;

top = top->next;

delete temp;

return value;

}

return '\0';

}
```

```
char peek() {
return (top != nullptr) ? top->data : '\0';
}

bool isEmpty() {
return top == nullptr;
}
};

int precedence(char op) {
if (op == '+' || op == '-') return 1;
if (op == '*' || op == '/') return 2;
if (op == '^') return 3;
return 0;
}
```

```
string infixToPostfix(const string& infix) {

Stack stack;

string postfix;



for (char ch : infix) {

if (isalnum(ch)) {



postfix += ch;

} else if (ch == '(') {

stack.push(ch);

} else if (ch == ')') {



while (!stack.isEmpty() && stack.peek() != '(') {

postfix += stack.pop();
```

```
}

stack.pop();

} else {



while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(ch)) {

postfix += stack.pop();

}

stack.push(ch);

}

}







while (!stack.isEmpty()) {

postfix += stack.pop();

}
```

```
return postfix;

}



int main() {

string infix;

cout << "Enter an infix expression: ";

cin >> infix;



string postfix = infixToPostfix(infix);

cout << "Postfix expression: " << postfix << endl;



return 0;

}
```