```
#include <iostream>
#include <stack>
#include <string>
#include <vector>
#include <sstream>
#include <cctype>
#include <algorithm>

using namespace std;

struct Node {
   string value;
   Node* left;
   Node* right;

   Node(string val) : value(val), left(nullptr), right(nullptr) {}
};

bool isOperator(const string& str) {
   return str == "+" || str == "-" || str == "*" || str == "/";
}

bool isComparisonOperator(const string& str) {
   return str == ">" || str == "<" || str == ">=" || str == "<=" || str == "==" || str == "!=";
}

bool isAssignmentOperator(const string& str) {
   return str == "=";
}

bool isNumber(const string& str) {
   return !str.empty() && all_of(str.begin(), str.end(), ::isdigit);
}

bool isIdentifier(const string& str) {
   return !str.empty() && isalpha(str[0]);
}

string trim(const string& str) {
   size_t start = str.find_first_not_of(" \t");
   size_t end = str.find_last_not_of(" \t");
   return (start == string::npos || end == string::npos) ? "" : str.substr(start, end - start + 1);
}

Node* buildExpressionTree(istringstream& iss) {
   stack<Node*> nodes;
   stack<string> operators;
```

```
    string token;

    while (iss >> token) {
        if (isOperator(token) || isComparisonOperator(token) || isAssignmentOperator(token))
{
            while (!operators.empty() &&
                   (operators.top() == "*" || operators.top() == "/" ||
                    (token == "+" || token == "-") && (operators.top() == "+" || operators.top() == "-
") ||
                    (isComparisonOperator(token) && isComparisonOperator(operators.top())))) {
                if (nodes.size() < 2) {
                    cerr << "Error: Not enough operands for operator " << operators.top() << endl;
                    return nullptr;
                }

                string op = operators.top();
                operators.pop();

                Node* rightOperand = nodes.top();
                nodes.pop();

                Node* leftOperand = nodes.top();
                nodes.pop();

                Node* operatorNode = new Node(op);
                operatorNode->left = leftOperand;
                operatorNode->right = rightOperand;

                nodes.push(operatorNode);
            }
            operators.push(token);
        } else if (isIdentifier(token) || isNumber(token)) {
            Node* operandNode = new Node(token);
            nodes.push(operandNode);
        }
    }

    while (!operators.empty()) {
        if (nodes.size() < 2) {
            cerr << "Error: Not enough operands for operator " << operators.top() << endl;
            return nullptr;
        }

        string op = operators.top();
        operators.pop();

        Node* rightOperand = nodes.top();
```

```
        nodes.pop();

        Node* leftOperand = nodes.top();
        nodes.pop();

        Node* operatorNode = new Node(op);
        operatorNode->left = leftOperand;
        operatorNode->right = rightOperand;

        nodes.push(operatorNode);
    }

    return nodes.top();
}

Node* buildIfElseTree(const string& condition, const string& thenExpr) {
    Node* ifNode = new Node("if");

    istringstream conditionStream(condition);
    Node* comparisonNode = buildExpressionTree(conditionStream);

    istringstream thenStream(thenExpr);
    Node* thenNode = buildExpressionTree(thenStream);

    ifNode->left = comparisonNode;
    ifNode->right = thenNode;

    return ifNode;
}

void separateIfElseComponents(const string& ifElseStatement, string& condition, string&
thenExpr) {
    size_t ifPos = ifElseStatement.find("if (");
    if (ifPos == string::npos) {
        cerr << "Invalid if-else statement format" << endl;
        return;
    }

    // Extract the condition
    size_t conditionStart = ifPos + 4; // Skip "if ("
    size_t conditionEnd = ifElseStatement.find(")", conditionStart);
    if (conditionEnd == string::npos) {
        cerr << "Missing closing parenthesis for condition" << endl;
        return;
    }
    condition = trim(ifElseStatement.substr(conditionStart, conditionEnd - conditionStart));
```

```
    // Extract the then expression
    size_t thenStart = ifElseStatement.find("{", conditionEnd) + 1;
    size_t thenEnd = ifElseStatement.find("}", thenStart);
    if (thenStart == string::npos || thenEnd == string::npos) {
        cerr << "Missing curly braces for 'then' expression" << endl;
        return;
    }
    thenExpr = trim(ifElseStatement.substr(thenStart, thenEnd - thenStart));
}

int treeHeight(Node* root) {
    if (!root) return 0;
    return 1 + max(treeHeight(root->left), treeHeight(root->right));
}

void fillTree(Node* root, vector<vector<string>>& matrix, int row, int col, int height, int
offset) {
    if (!root) return;

    if (row >= matrix.size() || col >= matrix[row].size() || col < 0) return;

    matrix[row][col] = root->value;

    if (root->left) {
        if (row + 1 < matrix.size() && col - offset / 2 >= 0) {
            matrix[row + 1][col - offset / 2] = "/";
            fillTree(root->left, matrix, row + 2, col - offset / 2, height, offset / 2);
        }
    }

    if (root->right) {
        if (row + 1 < matrix.size() && col + offset / 2 < matrix[row + 1].size()) {
            matrix[row + 1][col + offset / 2] = "\\";
            fillTree(root->right, matrix, row + 2, col + offset / 2, height, offset / 2);
        }
    }
}

void printTree(Node* root) {
    if (!root) return;

    int height = treeHeight(root);
    int width = (1 << (height - 1)) * 3;  // Width to accommodate nodes and connections

    vector<vector<string>> matrix(height * 2, vector<string>(width, " "));
    fillTree(root, matrix, 0, width / 2, height, width / 2);
```

```
    // Print the nodes
    for (const auto& row : matrix) {
        for (const auto& cell : row) {
            cout << cell;
        }
        cout << endl;
    }
}

int main() {
    // Example arithmetic expression
    string arithmeticExpression = "a = b + c";
    istringstream arithmeticStream(arithmeticExpression);

    // Build and print the arithmetic expression tree
    Node* arithmeticRoot = buildExpressionTree(arithmeticStream);
    if (arithmeticRoot) {
        cout << "Arithmetic expression tree:\n";
        printTree(arithmeticRoot);
    } else {
        cerr << "Failed to build arithmetic expression tree" << endl;
    }

    // Example if statement
    string ifElseStatement = "if (a > b) { a = b + c; }";
    string condition, thenExpr;

    separateIfElseComponents(ifElseStatement, condition, thenExpr);

    // Build and print the if expression tree
    Node* ifElseRoot = buildIfElseTree(condition, thenExpr);
    if (ifElseRoot) {
        cout << "If expression tree:\n";
        printTree(ifElseRoot);
    } else {
        cerr << "Failed to build if expression tree" << endl;
    }

    return 0;
}
```