```
#include <bits/stdc++.h>
using namespace std;

enum class TokenType {
    Keyword,
    Identifier,
    Operator,
    IntegerLiteral,
    Punctuation,
    Unknown
};

struct Token {
    TokenType type;
    string value;
};

string tokenTypeToString(TokenType type) {
    switch (type) {
        case TokenType::Keyword: return "Keyword";
        case TokenType::Identifier: return "Identifier";
        case TokenType::Operator: return "Operator";
        case TokenType::IntegerLiteral: return "Integer Literal";
        case TokenType::Punctuation: return "Punctuation";
        default: return "Unknown";
    }
}

vector<Token> lex(const string& code) {
```

```
vector<Token> tokens;

unordered_map<string, TokenType> keywords = {
    {"int", TokenType::Keyword},
    {"if", TokenType::Keyword},
    {"else", TokenType::Keyword},
    {"while", TokenType::Keyword},
    {"for", TokenType::Keyword},
    {"return", TokenType::Keyword},
    {"using", TokenType::Keyword},
    {"namespace", TokenType::Keyword}
};

unordered_map<string, TokenType> operators = {
    {"+", TokenType::Operator},
    {"-", TokenType::Operator},
    {"*", TokenType::Operator},
    {"/", TokenType::Operator},
    {"%", TokenType::Operator},
    {"==", TokenType::Operator}
};

regex identifierPattern(R"([a-zA-Z_][a-zA-Z0-9_]*)");
regex integerLiteralPattern(R"(\d+)");
regex operatorPattern(R"(\+|\-|\*|\/|%|==)");
regex punctuationPattern(R"([;,\{\}\(\)])");

istringstream stream(code);
string token;
```

```
    while (stream >> token) {
        if (keywords.find(token) != keywords.end()) {
            tokens.push_back({TokenType::Keyword, token});
        } else if (std::regex_match(token, identifierPattern)) {
            tokens.push_back({TokenType::Identifier, token});
        } else if (std::regex_match(token, integerLiteralPattern)) {
            tokens.push_back({TokenType::IntegerLiteral, token});
        } else if (std::regex_match(token, operatorPattern)) {
            tokens.push_back({TokenType::Operator, token});
        } else if (std::regex_match(token, punctuationPattern)) {
            tokens.push_back({TokenType::Punctuation, token});
        } else {
            tokens.push_back({TokenType::Unknown, token});
        }
    }

    return tokens;
}

int main() {
    string code = R"(
#include <iostream>
using namespace std;

int main() {
    cout << "Welcome";
    int x = 24 % 10;
    if (x == 4) {
```

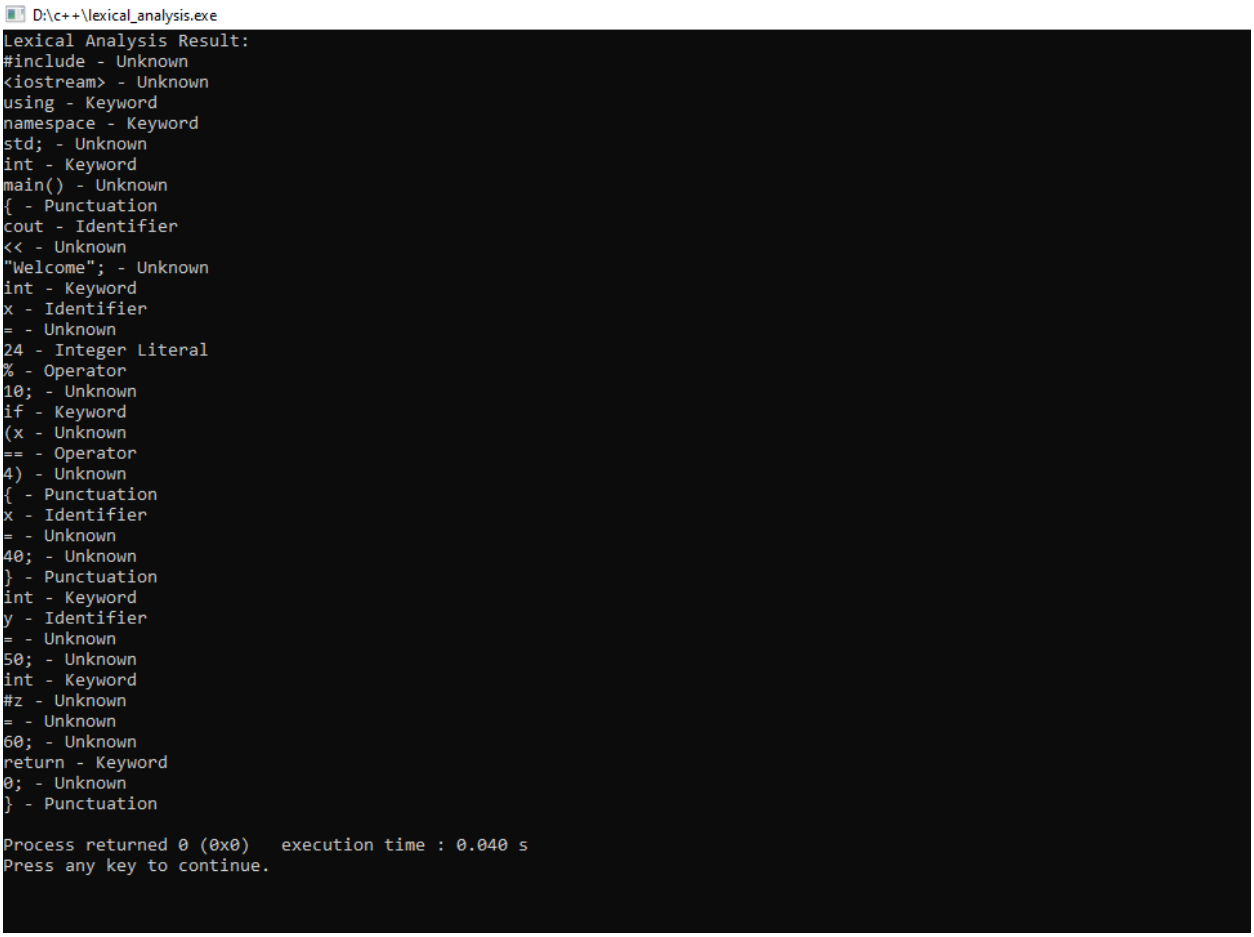
```
        x = 40;
    }
    int y = 50;
    int #z = 60;

    return 0;
}
)";
    auto tokens = lex(code);

    cout << "Lexical Analysis Result:\n";
    for (const auto& token : tokens) {
        cout << token.value << " - " << tokenTypeToString(token.type) << "\n";
    }

    return 0;
}
```

OUTPUT:


D:\c++\lexical_analysis.exe
Lexical Analysis Result:
#include - Unknown
<iostream> - Unknown
using - Keyword
namespace - Keyword
std; - Unknown
int - Keyword
main() - Unknown
{ - Punctuation
cout - Identifier
<< - Unknown
"Welcome"; - Unknown
int - Keyword
x - Identifier
= - Unknown
24 - Integer Literal
% - Operator
10; - Unknown
if - Keyword
(x - Unknown
== - Operator
4) - Unknown
{ - Punctuation
x - Identifier
= - Unknown
40; - Unknown
} - Punctuation
int - Keyword
y - Identifier
= - Unknown
50; - Unknown
int - Keyword
#z - Unknown
= - Unknown
60; - Unknown
return - Keyword
0; - Unknown
} - Punctuation
Process returned 0 (0x0) execution time : 0.040 s
Press any key to continue.