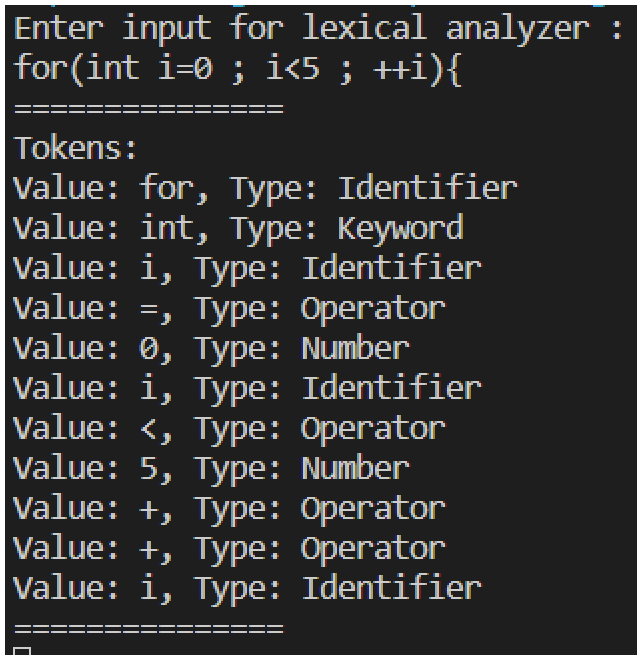
Q4.) Write a program to implement Lexical Analyser.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

enum class TokenType {
    Identifier, Keyword, Number, Operator, Unknown
};

struct Token {
    string value;
    TokenType type;
};

class LexicalAnalyzer {
private:
    vector<string> keywords = {"if", "else", "while", "do", "break", "continue", "int",
                               "double", "float", "return", "char", "case", "sizeof", "long",
                               "short", "typedef", "switch", "unsigned", "void", "static",
"struct", "goto"};

public:
    vector<Token> analyze(string &input) {
        vector<Token> tokens;
        string buffer;
        TokenType type = TokenType::Unknown;

        for (char ch : input) {
            if (isSpace(ch) || isDelimiter(ch)) {
                if (!buffer.empty()) {
                    type = determineType(buffer);
                    tokens.push_back({buffer, type});
                    buffer.clear();
                }
                if (isOperator(ch)) {
                    tokens.push_back({string(1, ch), TokenType::Operator});
                }
            } else {
                buffer += ch;
            }
        }
        if (!buffer.empty()) {
            type = determineType(buffer);
            tokens.push_back({buffer, type});
```

```cpp
        }

        return tokens;
    }

private:
    bool isSpace(char ch){
        return ch == ' ' || ch == '\t' || ch == '\n';
    }

    bool isDelimiter(char ch){
        string delimiters = " +-*/,;><=()[]{}";
        return delimiters.find(ch) != string::npos;
    }

    bool isOperator(char ch){
        string operators = "+-*/><=";
        return operators.find(ch) != string::npos;
    }

    bool isDigit(char ch){
        return ch >= '0' && ch <= '9';
    }

    bool isAlpha(char ch){
        return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
    }

    TokenType determineType(string &word){
        if (isDigit(word[0]) || (word[0] == '.' && word.size() > 1)) {
            return TokenType::Number;
        }

        if (isAlpha(word[0]) || word[0] == '_') {
            if (find(keywords.begin(), keywords.end(), word) != keywords.end()) {
                return TokenType::Keyword;
            } else {
                return TokenType::Identifier;
            }
        }

        return TokenType::Unknown;
    }
};

int main() {
    LexicalAnalyzer lexer;
    cout<<"Enter input for lexical analyzer : \n";
    while(true){
        string input;
        getline(cin, input);

        vector<Token> tokens = lexer.analyze(input);

        cout << "=============== \nTokens:\n";
        for (const auto &token : tokens) {
            cout << "Value: " << token.value << ", Type: ";
            switch (token.type) {
                case TokenType::Identifier: cout << "Identifier\n"; break;
                case TokenType::Keyword: cout << "Keyword\n"; break;
                case TokenType::Number: cout << "Number\n"; break;
                case TokenType::Operator: cout << "Operator\n"; break;
                default: cout << "Unknown\n";
            }
        }
        cout << "===============" <<endl;
    }

    return 0;
}
```

Output)



```
Enter input for lexical analyzer :
for(int i=0 ; i<5 ; ++i){
===============
Tokens:
Value: for, Type: Identifier
Value: int, Type: Keyword
Value: i, Type: Identifier
Value: =, Type: Operator
Value: 0, Type: Number
Value: i, Type: Identifier
Value: <, Type: Operator
Value: 5, Type: Number
Value: +, Type: Operator
Value: +, Type: Operator
Value: i, Type: Identifier
===============
```