**Name: Sparsh Karna**

**Reg Number: 23BDS1172**

**Lab Excercise: 4**

**Aim:**

The aim of this program is to perform lexical analysis on a given C++ source file and categorize tokens into keywords, literals, identifiers, and operators/punctuations.

**Procedure:**

1. To perform lexical analysis, the program first reads the input source file line by line, removes comments, and then splits each line into tokens using a regular expression. These tokens are then classified into four categories:
   1. Keywords → Predefined C++ reserved words (e.g., int, float, if, etc.) stored in a set for quick lookup.
   2. Operators/Punctuation → Symbols like +, -, {, }, ;, etc. stored in a set.
   3. Literals → Numbers, character constants, and string constants detected using regular expressions.
   4. Identifiers → User-defined names that follow valid naming conventions but are not keywords.
2. Each unique token found is stored in its respective set to avoid duplication. Finally, the program writes a formatted report of all token categories and their counts to an output file.

**Algorithm:**

1. Define sets for keywords and operators.
2. Create helper functions:
   * isLiteral(token) → Checks if a token is a numeric, char, or string literal.
   * isIdentifier(token) → Checks if a token matches the C++ identifier pattern and is not a keyword.
3. Define tokenize(line) to split a line into tokens using regex patterns for operators, identifiers, and literals.
4. Open the input source code file and output results file.
5. For each line in the input file:
   * Remove single-line comments.
   * Tokenize the line.
   * Classify each token into keywords, operators, literals, or identifiers.
6. Write the results to the output file, showing the count and list of each category.
7. Display a completion message on the console.

**Program:**

1. **Code:**

**#include <iostream>**

**#include <fstream>**

**#include <sstream>**

**#include <string>**

**#include <regex>**

**#include <unordered\_set>**

**#include <vector>**

**using namespace std;**

**const unordered\_set<string> reservedWords = {**

**"auto", "break", "case", "char", "const", "continue", "default", "do", "double",**

**"else", "enum", "extern", "float", "for", "goto", "if", "inline", "int", "long",**

**"register", "restrict", "return", "short", "signed", "sizeof", "static", "struct",**

**"switch", "typedef", "union", "unsigned", "void", "volatile", "while", "bool",**

**"namespace", "class", "public", "private", "protected", "true", "false", "using"};**

**const unordered\_set<string> operatorSymbols = {**

**";", ",", ".", "(", ")", "{", "}", "[", "]",**

**"=", "+", "-", "\*", "/", "<", ">", "!", "&", "|", ":",**

**"==", "<=", ">=", "!=", "++", "--", "+=", "-=", "\*=", "/="};**

**bool isLiteralToken(const string &tok)**

**{**

**static const regex numberPattern(R"(^\d+(\.\d+)?$)");**

**static const regex charPattern(R"(^'.'$)");**

**static const regex stringPattern(R"(^".\*"$)");**

**return regex\_match(tok, numberPattern) ||**

**regex\_match(tok, charPattern) ||**

**regex\_match(tok, stringPattern);**

**}**

**bool isValidIdentifier(const string &tok)**

**{**

**static const regex identPattern(R"(^[A-Za-z\_]\w\*$)");**

**return regex\_match(tok, identPattern) && !reservedWords.count(tok);**

**}**

**vector<string> splitTokens(const string &text)**

**{**

**vector<string> result;**

**static const regex tokenRegex(**

**R"((==|<=|>=|!=|\+\+|--|\+=|-=|\\*=|/=|[;,\.\(\)\{\}\[\]=+\\*/<>!&|:-]|\w+|".\*?"|'.'))");**

**for (auto it = sregex\_iterator(text.begin(), text.end(), tokenRegex);**

**it != sregex\_iterator(); ++it)**

**{**

**result.push\_back(it->str());**

**}**

**return result;**

**}**

**int main()**

**{**

**ifstream srcFile("input\_23BDS1172.txt");**

**ofstream outFile("output\_23BDS1172.txt");**

**if (!srcFile.is\_open())**

**{**

**cerr << "Error: Could not open source file.\n";**

**return 1;**

**}**

**unordered\_set<string> detectedKeywords, detectedLiterals, detectedIdentifiers, detectedOperators;**

**string line;**

**while (getline(srcFile, line))**

**{**

**// Strip single-line comments**

**line = regex\_replace(line, regex(R"(//.\*)"), "");**

**// Tokenize line**

**for (const string &tok : splitTokens(line))**

**{**

**if (reservedWords.count(tok))**

**{**

**detectedKeywords.insert(tok);**

**}**

**else if (operatorSymbols.count(tok))**

**{**

**detectedOperators.insert(tok);**

**}**

**else if (isLiteralToken(tok))**

**{**

**detectedLiterals.insert(tok);**

**}**

**else if (isValidIdentifier(tok))**

**{**

**detectedIdentifiers.insert(tok);**

**}**

**}**

**}**

**outFile << "=== Lexical Analysis Result ===\n\n";**

**outFile << "Keywords (" << detectedKeywords.size() << "):\n";**

**for (const auto &kw : detectedKeywords)**

**outFile << " - " << kw << "\n";**

**outFile << "\nLiterals (" << detectedLiterals.size() << "):\n";**

**for (const auto &lit : detectedLiterals)**

**outFile << " - " << lit << "\n";**

**outFile << "\nIdentifiers (" << detectedIdentifiers.size() << "):\n";**

**for (const auto &id : detectedIdentifiers)**

**outFile << " - " << id << "\n";**

**outFile << "\nOperators/Punctuations (" << detectedOperators.size() << "):\n";**

**for (const auto &op : detectedOperators)**

**outFile << " - " << op << "\n";**

**cout << "Lexical analysis completed. Results in 'output\_23BDS1172.txt'.\n";**

**return 0;**

**}**

1. **Input\_23BDS1172.txt:**

#include <iostream>

using namespace std;

int main() {

int a = 10;

float b = 3.14;

char c = 'x';

string msg = "Hello, tokens!";

// perform addition

float sum = a + b;

if (sum > 10) {

cout << msg << endl;

} else {

cout << "Sum is small" << endl;

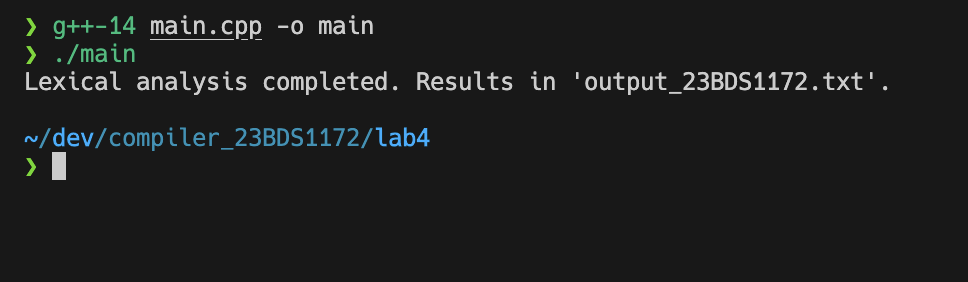
}

return 0;

}

**Output:**

**Terminal Output:**



**Output\_23BDS1172.txt:**

**=== Lexical Analysis Result ===**

**Keywords (8):**

**- return**

**- else**

**- if**

**- char**

**- float**

**- int**

**- namespace**

**- using**

**Literals (7):**

**- "Sum is small"**

**- "Hello, tokens!"**

**- 'x'**

**- 14**

**- 3**

**- 0**

**- 10**

**Identifiers (12):**

**- cout**

**- msg**

**- string**

**- sum**

**- c**

**- endl**

**- b**

**- a**

**- main**

**- std**

**- iostream**

**- include**

**Operators/Punctuations (10):**

**- }**

**- =**

**- (**

**- ;**

**- {**

**- )**

**- >**

**- +**

**- .**

**- <**

**Result:**

The program successfully reads a C++ source file, identifies all tokens, categorizes them into keywords, literals, identifiers, and operators/punctuation, and writes the result to an output file, demonstrating basic lexical analysis for a compiler front-end.