**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет кібернетики та комп'ютерних наук

**Звіт до лабораторної роботи №3**

з дисципліни “Системне програмування”

на тему “ Lexical Analysis”

Студента 3 курсу

групи ТТП-32

спеціальності 122 "Комп’ютерні науки"

Тимур КАТАСОНОВ

**Умова лабораторної роботи**

Реалізувати лексичний аналізатор мови програмування.

Розрізняти принаймні такі класи лексем:

* числа (десяткові, з плаваючою крапкою, шістнадцяткові),
* рядкові та символьні константи,
* директиви препроцесора,
* коментарі,
* зарезервовані слова,
* оператори,
* розділові знаки,
* ідентифікатори,
* інші, характерні для відповідної мови програмування.

Позначати ситуації з помилками (наприклад, нерозпізнавані символи).

Можливі варіанти виводу результату роботи програми:

* “розфарбування” тексту програми (наприклад, кольором),
* вивід пар < лексема , тип\_лексеми > послідовно для всього тексту програми.

Варіант №4: Мова програмування Java.

**Код програми**

#include <iostream>  
#include <fstream>  
#include <regex>  
#include <string>  
#include <unordered\_set>  
  
using namespace std;  
  
unordered\_set<string> keywords = **{** "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class", "const", "continue",  
 "default", "do", "double", "else", "enum", "extends", "final", "finally", "float", "for", "goto", "if",  
 "implements", "import", "instanceof", "int", "interface", "long", "native", "new", "package", "private",  
 "protected", "public", "return", "short", "static", "strictfp", "super", "switch", "synchronized",  
 "this", "throw", "throws", "transient", "try", "void", "volatile", "while", "true", "false", "null"  
**}**;  
  
unordered\_set<string> operators = **{** "+", "-", "\*", "/", "%", "++", "--", "==", "!=", ">", "<", ">=", "<=", "&&", "||", "!", "=", "+=", "-=", "\*=", "/="  
**}**;  
  
unordered\_set<char> delimiters = **{** '(', ')', '{', '}', '[', ']', ';', ',', '.'  
**}**;  
  
string ClassifyToken(string &token) {  
 static bool inMultilineComment = false;  
  
 if (inMultilineComment) {  
 size\_t endCommentPos = token.find("\*/");  
 if (endCommentPos != std::string::npos) {  
 inMultilineComment = false;  
 }  
 return "Multiline comment";  
 } else {  
 size\_t startCommentPos = token.find("/\*");  
 if (startCommentPos != std::string::npos) {  
 inMultilineComment = true;  
 size\_t endCommentPos = token.find("\*/");  
 if (endCommentPos != std::string::npos) {  
 inMultilineComment = false;  
 }  
 return "Multiline comment";  
 } else {  
 if (regex\_match(token, regex("//.\*"))) {  
 return "Inline comment";  
 }  
  
 if (regex\_match(token, regex("\".\*\""))) {  
 return "String Literal";  
 } else if (regex\_match(token, regex("'.'"))) {  
 return "Character Literal";  
 }  
  
 if (regex\_match(token, regex("[0-9]+"))) {  
 return "Integer";  
 } else if (regex\_match(token, regex("[0-9]\*\\.[0-9]+"))) {  
 return "Float";  
 } else if (regex\_match(token, regex("0[xX][0-9a-fA-F]+"))) {  
 return "Hexadecimal";  
 }  
  
 if (keywords.find(token) != keywords.end()) {  
 return "Keyword";  
 }  
  
 if (operators.find(token) != operators.end()) {  
 return "Operator";  
 }  
  
 if (token.length() == 1 && delimiters.find(token[0]) != delimiters.end()) {  
 return "Delimiter";  
 }  
  
 if (regex\_match(token, regex("[a-zA-Z\_][a-zA-Z0-9\_]\*"))) {  
 return "Identifier";  
 }  
  
 return "Unknown";  
 }  
 }  
}  
  
void LexicalAnalysis(string &inputFileName, string &outputFileName) {  
 ifstream inputFile(inputFileName);  
 ofstream outputFile(outputFileName);  
  
 if (!inputFile.is\_open()) {  
 cerr << "Error: Unable to open input file." << endl;  
 return;  
 }  
  
 if (!outputFile.is\_open()) {  
 cerr << "Error: Unable to open output file." << endl;  
 return;  
 }  
  
 string line;  
 regex tokenRegex("/\\\*|\\\*/|//.\*|\".\*?\"|'.'|[a-zA-Z\_][a-zA-Z0-9\_]\*|[0-9]\*\\.[0-9]+|[0-9]+|0[xX][0-9a-fA-F]+|[+\\-\*/%=!&|<>]=?|[(){},.;]");  
 while (getline(inputFile, line)) {  
 auto tokensBegin = sregex\_iterator(line.begin(), line.end(), tokenRegex);  
 auto tokensEnd = sregex\_iterator();  
  
 for (sregex\_iterator it = tokensBegin; it != tokensEnd; it++) {  
 string token = it->str();  
 outputFile << ClassifyToken(token) << ":\t" << token << endl;  
 }  
 }  
  
 inputFile.close();  
 outputFile.close();  
}  
  
int main() {  
 string inputFileName1 = R"(D:\Homeworks\sp\Lab3\Input\input1.java)";  
 string inputFileName2 = R"(D:\Homeworks\sp\Lab3\Input\input2.java)";  
 string inputFileName3 = R"(D:\Homeworks\sp\Lab3\Input\input3.java)";  
 string outputFileName1 = R"(D:\Homeworks\sp\Lab3\Output\output1.txt)";  
 string outputFileName2 = R"(D:\Homeworks\sp\Lab3\Output\output2.txt)";  
 string outputFileName3 = R"(D:\Homeworks\sp\Lab3\Output\output3.txt)";  
  
 LexicalAnalysis(inputFileName1, outputFileName1);  
 LexicalAnalysis(inputFileName2, outputFileName2);  
 LexicalAnalysis(inputFileName3, outputFileName3);  
  
 return 0;  
}

**Конкретніше розберемо код:**

1. ClassifyToken(string &token) – отримує токен, та визначає якому з класів він належить (в такій самій послідовності):
   1. Multiple comment
   2. Inline comment
   3. String literal
   4. Character literal
   5. Integer
   6. Float
   7. Hexadecimal
   8. Keyword
   9. Operator
   10. Delimiter
   11. Identifier
   12. Unknown.
2. LexicalAnalysis(string &inputFileName, string &outputFileName) – отримує input та output файли, читає з вхідного файлу програму порядково, розбиває його на токени, базуючись на регулярному виразі: "/\\\*|\\\*/|//.\*|\".\*?\"|'.'|[a-zA-Z\_][a-zA-Z0-9\_]\*|[0-9]\*\\.[0-9]+|[0-9]+|0[xX][0-9a-fA-F]+|[+\\-\*/%=!&|<>]=?|[(){},.;]", де:
   1. /\\\*|\\\*/ – еквівалент /\* та \*/, тобто границі багаторядкового коментаря
   2. //.\* – однорядковий коментар
   3. \".\*?\" – рядковий літерал
   4. '.' – символьний літерал
   5. [a-zA-Z\_][a-zA-Z0-9\_]\* - ідентифікатор
   6. [0-9]\*\\.[0-9]+ - дробове число
   7. [0-9]+ - ціле число
   8. 0[xX][0-9a-fA-F]+ - шістнадцяткове число
   9. [+\\-\*/%=!&|<>]=? – оператори
   10. [(){},.;] – розділові знаки
3. int main() – запускає програму, викликає LexicalAnalysis для кожного тестового файлу.