ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ № 4

за курсом “Мовні технології”

студента групи ПЗ-24у-1

Кондрачука Олександра Вадимовича

2024/2025 н.р.

1. **Постановка задачі**

Реалізувати лексичний аналізатор JavaScript.

1. **Опис розв’язку**

Код виконує лексичний аналіз введеного коду, розпізнаючи різні типи лексем, такі як ключові слова, ідентифікатори, числа, рядкові та символьні константи, оператори та коментарі. Розглянемо кожну частину коду текстом.

Опис розв’язку

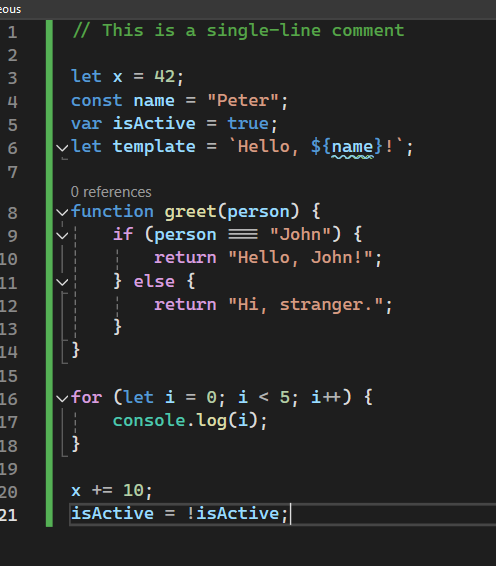
1. Ініціалізація:
   * Функція isKeyword використовує статичний контейнер unordered\_set для зберігання ключових слів.
   * У функції analyze визначається перелік станів автомату (State), які використовуються для розпізнавання різних типів лексем: початковий стан (Start), ідентифікатор (Identifier), число (Number), рядкова константа (String), символьна константа, оператор (Operator), коментар (LineComment) та інші.
2. Обробка символів:
   * У початковому стані (Start) кожен символ перевіряється, щоб визначити, який тип лексеми починається. Якщо це літера або підкреслення, то починається ідентифікатор, якщо цифра — число, якщо лапки — рядкова або символьна константа, якщо спеціальний символ — оператор або розділювач.
   * Коли автомат переходить у відповідний стан (наприклад, Identifier для ідентифікаторів), він продовжує додавати символи до поточної лексеми доти, доки не буде знайдено символ, який не належить цьому типу лексеми. Потім лексема завершується і обробляється відповідним чином (наприклад, перевіряється, чи є ідентифікатор ключовим словом).
   * Якщо автомат перебуває в стані коментаря (LineComment), він додає всі символи до коментаря доти, доки не зустріне символ нового рядка, після чого коментар виводиться.
3. Розпізнавання та вивід:
   * Після завершення зчитування кожної лексеми автомат повертається до початкового стану (Start), готовий до обробки наступної лексеми.
   * Лексеми різних типів виводяться на екран з відповідним позначенням: ключові слова, ідентифікатори, числа, рядкові константи, символьні константи, оператори, коментарі та розділювачі.
4. У функції main файл example.js відкривається для зчитування. Вміст файлу зчитується пострічково та додається до рядка code.
5. Після завершення зчитування файл закривається, а функція analyze викликається для аналізу коду.

**3. Опис інтерфейсу**

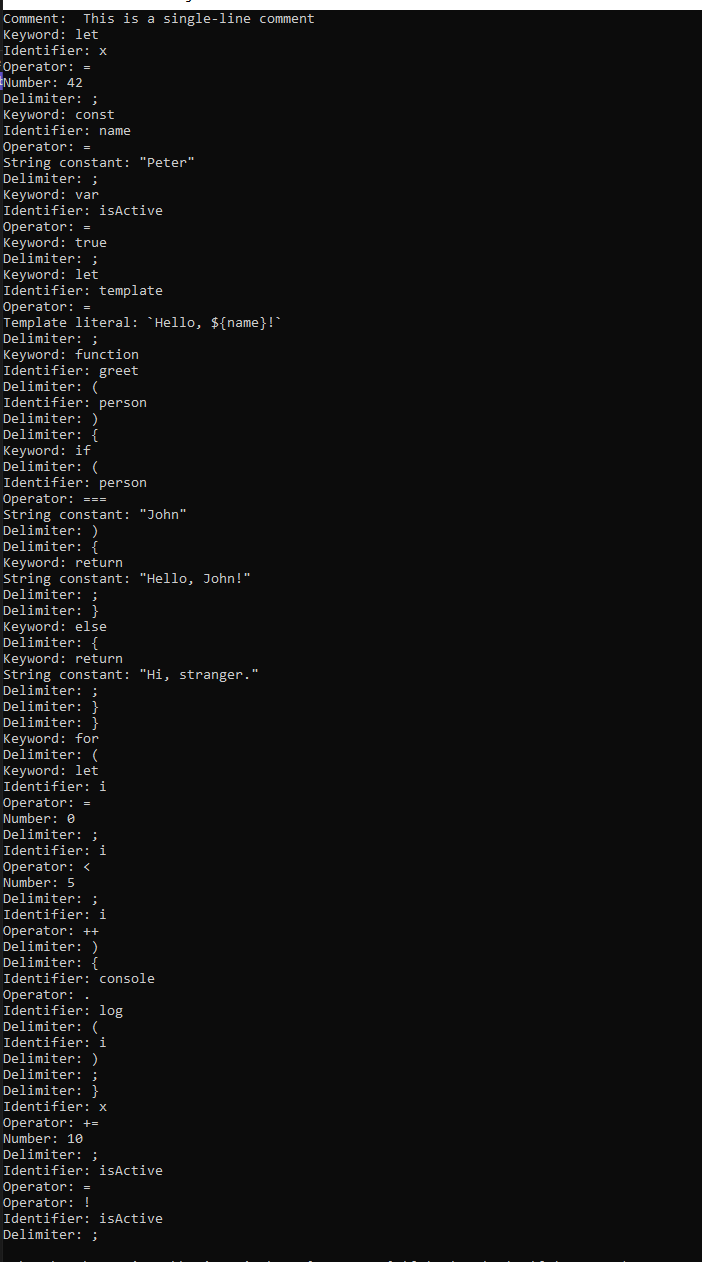
Створити файл з кодом JavaScript.

**4. Тестування програми**

**Вхідний файл:**

****

**Фрагмент результату:**

****

**5. Аналіз помилок**

**немає**

**6. додаток:**

#include <iostream>

#include <fstream>

#include <string>

#include <cctype>

#include <unordered\_set>

enum class State

{

Start,

Identifier,

Number,

String,

TemplateLiteral,

Operator,

LineComment,

BlockComment

};

static const std::unordered\_set<std::string> keywords =

{

"if", "else", "for", "while", "do", "break", "continue",

"switch", "case", "default", "function", "return", "var", "let", "const",

"new", "this", "typeof", "instanceof", "void", "delete", "try", "catch",

"finally", "throw", "class", "extends", "super", "import", "from", "export",

"in", "of", "with", "yield", "await", "async", "true", "false", "null", "undefined"

};

bool isKeyword(const std::string& word)

{

return keywords.count(word) > 0;

}

std::string readFile(const std::string& filename)

{

std::ifstream file(filename);

if (!file)

{

throw std::runtime\_error("Failed to open file: " + filename);

}

std::string content, line;

while (std::getline(file, line))

{

content += line + '\n';

}

return content;

}

void outputToken(const std::string& lexeme, State state)

{

if (lexeme.empty())

{

return;

}

switch (state)

{

case State::Identifier:

std::cout << (isKeyword(lexeme) ? "Keyword: " : "Identifier: ") << lexeme << '\n';

break;

case State::Number:

std::cout << "Number: " << lexeme << '\n';

break;

case State::String:

std::cout << "String constant: " << lexeme << '\n';

break;

case State::TemplateLiteral:

std::cout << "Template literal: " << lexeme << '\n';

break;

case State::Operator:

std::cout << "Operator: " << lexeme << '\n';

break;

default:

break;

}

}

void analyze(const std::string& code)

{

State state = State::Start;

std::string lexeme;

std::string comment;

char quote = '\0';

for (size\_t i = 0; i < code.size(); ++i)

{

char c = code[i];

char next = (i + 1 < code.size()) ? code[i + 1] : '\0';

switch (state)

{

case State::Start:

if (std::isalpha(c) || c == '\_')

{

state = State::Identifier;

lexeme += c;

}

else if (std::isdigit(c))

{

state = State::Number;

lexeme += c;

}

else if (c == '\'' || c == '"')

{

state = State::String;

quote = c;

lexeme += c;

}

else if (c == '`') {

state = State::TemplateLiteral;

quote = c;

lexeme += c;

}

else if (c == '/' && next == '/')

{

state = State::LineComment;

++i;

comment.clear();

}

else if (c == '/' && next == '\*')

{

state = State::BlockComment;

++i;

comment.clear();

}

else if (std::ispunct(c) && c != ';' && c != '(' && c != ')' && c != '[' && c != ']' && c != '{' && c != '}')

{

state = State::Operator;

lexeme += c;

}

else if (!std::isspace(c))

{

std::cout << "Delimiter: " << c << '\n';

}

break;

case State::Identifier:

if (std::isalnum(c) || c == '\_')

{

lexeme += c;

}

else

{

outputToken(lexeme, state);

lexeme.clear();

state = State::Start;

--i;

}

break;

case State::Number:

if (std::isdigit(c) || c == '.')

{

lexeme += c;

}

else

{

outputToken(lexeme, state);

lexeme.clear();

state = State::Start;

--i;

}

break;

case State::String:

lexeme += c;

if (c == quote && lexeme[lexeme.size() - 2] != '\\')

{

outputToken(lexeme, state);

lexeme.clear();

state = State::Start;

}

break;

case State::TemplateLiteral:

lexeme += c;

if (c == '`' && lexeme[lexeme.size() - 2] != '\\')

{

outputToken(lexeme, state);

lexeme.clear();

state = State::Start;

}

break;

case State::Operator:

if (c == '=' || c == '&' || c == '|' || c == '+' || c == '-' || c == '>')

{

lexeme += c;

}

else

{

outputToken(lexeme, state);

lexeme.clear();

state = State::Start;

--i;

}

break;

case State::LineComment:

if (c == '\n')

{

std::cout << "Comment: " << comment << '\n';

state = State::Start;

}

else

{

comment += c;

}

break;

case State::BlockComment:

if (c == '\*' && next == '/')

{

++i;

std::cout << "Multiline Comment: " << comment << '\n';

state = State::Start;

}

else

{

comment += c;

}

break;

}

}

outputToken(lexeme, state);

}

int main()

{

try

{

std::string code = readFile("example.js");

analyze(code);

}

catch (const std::exception& ex)

{

std::cerr << ex.what() << '\n';

return 1;

}

return 0;

}