ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

Проектирование лексического анализатора

Выполнил: Тимощенко Т.С.

Вариант - 13

ЗАДАНИЕ ПО ЛАБОРАТОРНОЙ РАБОТЕ

По условию задания необходимо создать лексический анализатор (сканер), который:

1. Считывает входной текст из символьного файла, в котором записаны несколько арифметических выражений, разделенных символом ; (точка с запятой).
2. Распознает в тексте следующие типы лексем:
   * **идентификаторы** (последовательность латинских букв и/или цифр, начинающаяся с буквы или символа подчёркивания, длиной не более 32 символов);
   * **символьные константы** (ровно один символ в одинарных кавычках);
   * **знак присваивания** :=;
   * **знаки операций** +, -, \*, /;
   * **круглые скобки** ( и );
   * **точка с запятой** ; как разделитель выражений;
   * **однострочные комментарии** (например, начинающиеся с // и идущие до конца строки);
3. Игнорирует пробелы, табуляцию и переводы строк;
4. При нахождении недопустимых символов или неправильных форм лексем (например, неправильная символьная константа, слишком длинный идентификатор и т. п.) формирует лексему типа "Ошибка";
5. Выводит в итоговую таблицу лексем тип и значение каждой лексемы и дополнительно сообщает общее количество лексем и количество ошибок.

ОПИСАНИЕ КС-ГРАММАТИКИ ВХОДНОГО ЯЗЫКА

Опишем входной язык в форме Бэкуса-Наура (BNF). В нашем случае язык состоит из последовательности **арифметических выражений**, разделенных точкой с запятой:

<Program> ::= <Expr> | <Expr> ";" <Program>

<Expr> ::= <AssignExpr>

<AssignExpr> ::= <Identifier> ":=" <ArithmeticExpr>

<ArithmeticExpr> ::= <Term> | <Term> { ("+" | "-") <Term> }

<Term> ::= <Factor> | <Factor> { ("\*" | "/") <Factor> }

<Factor> ::= <Identifier> | <CharConst> | "(" <ArithmeticExpr> ")"

<Identifier> ::= <LetterOrUnderscore> { <LetterOrDigitOrUnderscore> }

<CharConst> ::= "'" <AnyChar> "'"

<LetterOrUnderscore> ::= (A..Z | a..z | "\_")

<LetterOrDigitOrUnderscore> ::= (A..Z | a..z | 0..9 | "\_")

Здесь:

* <Identifier> описывает идентификатор,
* <CharConst> — символьную константу,
* В выражениях допускаются операторы +, -, \*, / и круглые скобки.
* ; используется как разделитель выражений.
* Дополнительно, по заданию, допускаются комментарии, которые в грамматике не отражены явно, но игнорируются в тексте.

ОПИСАНИЕ АЛГОРИТМА РАБОТЫ СКАНЕРА

Алгоритм лексического анализа можно описать следующим образом:

1. **Инициализация**: программа считывает содержимое входного файла в строку.
2. **Основной цикл**: последовательно просматриваем символы входной строки, используя переменную-счётчик index.
   * **Пропуск пробелов и переводов строк**: если текущий символ — это whitespace, инкрементируем index.
   * **Проверка комментариев**: если встретили последовательность //, пропускаем все символы до конца текущей строки.
   * **Проверка символьной константы** 'x': проверяем, что между ' и ' ровно один символ.
   * **Присваивание** :=: если встретили :, смотрим следующий символ, ожидаем =.
   * **Операторы** +, -, \*, /: один символ.
   * **Круглые скобки** (, ): один символ.
   * **Точка с запятой** ;: один символ.
   * **Идентификатор**: начинается с буквы/подчёркивания, продолжается буквами/цифрами/подчёркиваниями, ограничен 32 символами.
   * **Ошибка**: если символ не подходит ни под одно из правил, фиксируем лексему "Ошибка".
3. **Формирование таблицы**: на каждую обнаруженную лексему создаётся запись в таблице (списке), содержащая:
   1. Номер лексемы,
   2. Тип лексемы (Идентификатор, Оператор, Скобка, Символьная константа, Ошибка и т. п.),
   3. Строковое значение лексемы.
4. **Подсчёт ошибок**: при формировании лексемы типа "Ошибка" увеличивается счётчик ошибок.
5. **Вывод результатов**:
   * Таблица лексем отображается в интерфейсе.
   * После завершения анализа выводится общее количество лексем и количество ошибок.

ТЕКСТ ПРОГРАММЫ

**4.1. ChooseFile\_Click**

Эта функция отвечает за выбор файла через стандартный диалог. Пользователь выбирает файл с расширением .txt, после чего путь к файлу отображается в текстовом поле.

private void ChooseFile\_Click(object sender, RoutedEventArgs e)

{

var openFileDialog = new OpenFileDialog

{

Filter = "Текстовые файлы|\*.txt|Все файлы|\*.\*"

};

if (openFileDialog.ShowDialog() == true)

{

FileNameTextBox.Text = openFileDialog.FileName;

}

}

**4.2. LoadFile\_Click**

Считывает содержимое выбранного текстового файла и отображает его в поле ввода. После этого запускается лексический анализ с помощью функции Tokenize, и результаты отображаются в виде таблицы.

private void LoadFile\_Click(object sender, RoutedEventArgs e)

{

var filePath = FileNameTextBox.Text;

if (!string.IsNullOrWhiteSpace(filePath) && File.Exists(filePath))

{

string text = File.ReadAllText(filePath);

InputTextBox.Text = text;

var tokens = Tokenize(text);

TokensDataGrid.ItemsSource = tokens;

}

else

{

MessageBox.Show("Файл не найден или путь к файлу не указан.");

}

}

**4.3. ExitButton\_Click**

Закрывает приложение по нажатию кнопки.

private void ExitButton\_Click(object sender, RoutedEventArgs e)

{

this.Close();

}

**4.4. Tokenize**

Основная функция, реализующая лексический анализ. Обрабатывает строку посимвольно и формирует список лексем в соответствии с правилами языка. Также считает количество ошибок и отображает общее количество лексем и количество ошибок в сообщении.

private List<Lexeme> Tokenize(string input)

{

var result = new List<Lexeme>();

int index = 0;

int tokenCounter = 1;

int errorCount = 0;

while (index < input.Length)

{

char c = input[index];

// Пропуск пробелов, табуляций, переводов строк

if (char.IsWhiteSpace(c))

{

index++;

continue;

}

// Однострочные комментарии //

if (c == '/' && index + 1 < input.Length && input[index + 1] == '/')

{

index += 2;

while (index < input.Length && input[index] != '\n')

{

index++;

}

continue;

}

// Символьная константа: 'x'

if (c == '\'')

{

if (index + 2 < input.Length && input[index + 2] == '\'')

{

string charConst = input.Substring(index, 3);

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Символьная константа",

LexemeValue = charConst

});

index += 3;

}

else

{

errorCount++;

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Ошибка",

LexemeValue = "Некорректная символьная константа"

});

index++;

}

continue;

}

// Присваивание: :=

if (c == ':' && index + 1 < input.Length && input[index + 1] == '=')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Присваивание",

LexemeValue = ":="

});

index += 2;

continue;

}

// Операторы +, -, \*, /

if (c == '+' || c == '-' || c == '\*' || c == '/')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Оператор",

LexemeValue = c.ToString()

});

index++;

continue;

}

// Круглые скобки ( )

if (c == '(' || c == ')')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Скобка",

LexemeValue = c.ToString()

});

index++;

continue;

}

// Точка с запятой ;

if (c == ';')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Разделитель",

LexemeValue = ";"

});

index++;

continue;

}

// Идентификатор: [a-zA-Z\_][a-zA-Z0-9\_]\* (не более 32 символов)

if (IsIdentifierStart(c))

{

int startPos = index;

index++;

while (index < input.Length && IsIdentifierPart(input[index]))

{

index++;

}

string ident = input.Substring(startPos, index - startPos);

if (ident.Length <= 32)

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Идентификатор",

LexemeValue = ident

});

}

else

{

errorCount++;

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Ошибка",

LexemeValue = $"Идентификатор '{ident}' превышает 32 символа"

});

}

continue;

}

// Всё остальное -> "Ошибка"

errorCount++;

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Ошибка",

LexemeValue = c.ToString()

});

index++;

}

MessageBox.Show(

$"Обнаружено ошибок: {errorCount}\nВсего лексем: {result.Count}",

"Лексический анализ"

);

return result;

}

**4.5. IsIdentifierStart**

Проверяет, может ли символ быть началом идентификатора (буква или подчёркивание).

private bool IsIdentifierStart(char c)

{

return char.IsLetter(c) || c == '\_';

}

**4.6. IsIdentifierPart**

Проверяет, может ли символ входить в идентификатор (буква, цифра или подчёркивание).

private bool IsIdentifierPart(char c)

{

return char.IsLetterOrDigit(c) || c == '\_';

}

ВЫВОДЫ ПО ПРОДЕЛАННОЙ РАБОТЕ

В ходе выполнения лабораторной работы реализован сканер, который:

1. Читает текст из входного файла и выводит его на экран.
2. Распознаёт заданные лексемы (идентификаторы, символьные константы, :=, +, -, \*, /, круглые скобки, ;) в соответствии с заданием.
3. Игнорирует комментарии в стиле //.
4. Ограничивает длину идентификатора 32 символами.
5. Фиксирует все некорректные ситуации (слишком длинный идентификатор, лишние символы в символьной константе и др.) как лексемы типа «Ошибка».
6. Подсчитывает общее количество лексем и количество ошибок, выводит данную статистику.

Таким образом, программа успешно решает поставленную задачу лексического анализа простейшего языка арифметических выражений с символьными константами, операторами, присваиванием, идентификаторами и комментариями, и формирует таблицу лексем. На выходе программа предоставляет удобную визуализацию результата.

ПРИЛОЖЕНИЕ

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows;

namespace lab2

{

public partial class MainWindow : Window

{

public class Lexeme

{

public int Number { get; set; }

public string LexemeType { get; set; }

public string LexemeValue { get; set; }

}

public MainWindow()

{

InitializeComponent();

}

private void ChooseFile\_Click(object sender, RoutedEventArgs e)

{

var openFileDialog = new OpenFileDialog

{

Filter = "Текстовые файлы|\*.txt|Все файлы|\*.\*"

};

if (openFileDialog.ShowDialog() == true)

{

FileNameTextBox.Text = openFileDialog.FileName;

}

}

private void LoadFile\_Click(object sender, RoutedEventArgs e)

{

var filePath = FileNameTextBox.Text;

if (!string.IsNullOrWhiteSpace(filePath) && File.Exists(filePath))

{

string text = File.ReadAllText(filePath);

InputTextBox.Text = text;

var tokens = Tokenize(text);

TokensDataGrid.ItemsSource = tokens;

}

else

{

MessageBox.Show("Файл не найден или путь к файлу не указан.");

}

}

private void ExitButton\_Click(object sender, RoutedEventArgs e)

{

this.Close();

}

private List<Lexeme> Tokenize(string input)

{

var result = new List<Lexeme>();

int index = 0;

int tokenCounter = 1;

int errorCount = 0;

while (index < input.Length)

{

char c = input[index];

// Пропуск пробелов, табуляций, переводов строк

if (char.IsWhiteSpace(c))

{

index++;

continue;

}

// Однострочные комментарии //

if (c == '/' && index + 1 < input.Length && input[index + 1] == '/')

{

index += 2;

while (index < input.Length && input[index] != '\n')

{

index++;

}

continue;

}

// Символьная константа: 'x'

if (c == '\'')

{

if (index + 2 < input.Length && input[index + 2] == '\'')

{

string charConst = input.Substring(index, 3);

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Символьная константа",

LexemeValue = charConst

});

index += 3;

}

else

{

errorCount++;

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Ошибка",

LexemeValue = "Некорректная символьная константа"

});

index++;

}

continue;

}

// Присваивание: :=

if (c == ':' && index + 1 < input.Length && input[index + 1] == '=')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Присваивание",

LexemeValue = ":="

});

index += 2;

continue;

}

// Операторы +, -, \*, /

if (c == '+' || c == '-' || c == '\*' || c == '/')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Оператор",

LexemeValue = c.ToString()

});

index++;

continue;

}

// Круглые скобки ( )

if (c == '(' || c == ')')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Скобка",

LexemeValue = c.ToString()

});

index++;

continue;

}

// Точка с запятой ;

if (c == ';')

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Разделитель",

LexemeValue = ";"

});

index++;

continue;

}

// Идентификатор: [a-zA-Z\_][a-zA-Z0-9\_]\* (не более 32 символов)

if (IsIdentifierStart(c))

{

int startPos = index;

index++;

while (index < input.Length && IsIdentifierPart(input[index]))

{

index++;

}

string ident = input.Substring(startPos, index - startPos);

if (ident.Length <= 32)

{

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Идентификатор",

LexemeValue = ident

});

}

else

{

errorCount++;

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Ошибка",

LexemeValue = $"Идентификатор '{ident}' превышает 32 символа"

});

}

continue;

}

// Всё остальное -> "Ошибка"

errorCount++;

result.Add(new Lexeme

{

Number = tokenCounter++,

LexemeType = "Ошибка",

LexemeValue = c.ToString()

});

index++;

}

MessageBox.Show(

$"Обнаружено ошибок: {errorCount}\nВсего лексем: {result.Count}",

"Лексический анализ"

);

return result;

}

private bool IsIdentifierStart(char c)

{

return char.IsLetter(c) || c == '\_';

}

private bool IsIdentifierPart(char c)

{

return char.IsLetterOrDigit(c) || c == '\_';

}

}

}