высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Энгельсский технологический институт

Кафедра Естественные и математические науки

Специальность 09.03.04 Программная инженерия

Отчёт по Лабораторной работе №2 по дисциплине “Теория языков программирования и методы трансляции”

|  |  |
| --- | --- |
|  | Выполнил: студент 3 курса  учебной группы ПИНЖ-31  очной формы обучения  Панасенков М.Н. |

Энгельс 2023

**2.Вариант № 13**

**Условие:**

1.Получить вариант задания у преподавателя.

2. В соответствии с выданным вариантом выполнить следующее:

2.1. Составит техническое задание (ТЗ) на разработку программы сканера, производящей лексический анализ произвольных текстов в пределах установленного алфавита.

2.2. Согласовать ТЗ с преподавателем.

2.3. Разработать программу-сканер на языках Паскаль, С++ или в интегрированных средах по собственному усмотрению.

2.4. Провести тестирование и отладку программы (предусмотреть все случаи вывода сообщений об ошибках пользователю).

2.5. Составить отчёт по работе и приложить к нему ТЗ.

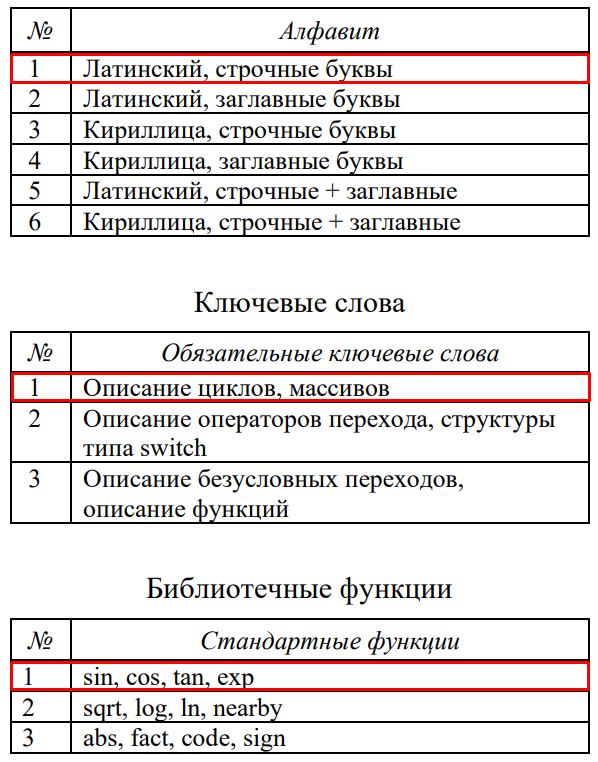


Рисунок 1.Изначальные данные для анализатора

**Выбранные ключевые слова:**

"program", "var", "begin", "end", "if", "then", "else", "for", "to", "do", "sin", "cos", "tan", "exp", "integer", "real", "const", "array", "of", "Writeln"

**Выбранные стандартные функции:**

sin, cos, tan, exp

**3.Внутренние таблицы сканера:**



Рисунок 2.Таблица ключевых слов и разделителей

**4.Техническое задание**

**Основания для разработки**

Наименование документа: 2\_5280506778256160813

Организация утвердивший этот документ: ЭТИ СГТУ

Наименование разработки: Программы (сканера)

**Назначение разработки.**

**Требования к программе или программному изделию.**

**Требования к функциональным характеристикам:**

Программа должна производить лексический анализ входного текста для транслятора:

* создание таблиц выделенных лексем для конечных классов;
* кодирование идентификаторов, разделителей и констант;
* проверка правильности написания ключевых слов, операторов, стандартных функций и использование служебных символов;
* отображение дескрипторного текста.

**Требования к надежности:**

Программа должна стабильно обрабатывать текстовую информацию и производить лексический анализ.

**Условия эксплуатации:**

При использование программы пользователь должен облать минимальными навыками работы с языком программирования Python.

**Требования к информационно и программной совместимости:**

Программа должна быть запущенна на устройстве соответствующем минимальным требованиям языка Python:

* 64- или 32-разрядный процессор на архитектуре Intel или AMD;
* ГБ оперативной памяти, желательно 4 ГБ;
* 5 ГБ свободного места в памяти компьютера;
* Windows, MacOS, Linux

**Требования к программной документации.**

В комплекте с программой должен присутствовать документ со следующим содержимым:

* Код программы
* Вводимые текстовые данные
* Вывод программы

Стадии и этапы разработки

**Стадии и их этапы:**

Разработка технического задания:

1. Составление требования
2. Прописывание функций

Разработка программы:

1. Создание таблиц Ключевые слова и Разделители
2. Написание проверки на Идентификаторы и Константы
3. Вывод полученных таблиц (Ключевые слова, Разделители, Идентификаторы, Константы).
4. Написание алгоритма преобразование текста в дескрипторный текст
5. Вывод дескрипторного текста

Написание документа “Отчёт”:

1. Прописывание условий задачи
2. Вывод листинга программы
3. Вывод отладочного примера работа программы

**Сроки разработки:**

Дата начала разработки: 30.10.23

Дата завершения разработки:31.10.23

|  |  |
| --- | --- |
| Стадии: | Затраченное время |
| Разработка технического задания | 2 часа |
| Разработка программы | 2 часа |
| Написание документа “Отчёт”: | 1 час |

**Исполнители:**

Панасенков Максим Николаевич

**Порядок контроля и приёмки**

**5.Листинг программы**

**Листинг программы:**

namespace TeorYazikovLaba2

{

class Program

{

static string GenerateDescriptor(string code, LexicalAnalyzer analyzer)

{

string descriptorText = "";

string[] lines = code.Split(new string[] { "\r\n", "\n" }, StringSplitOptions.RemoveEmptyEntries);

int lineIndex = 0;

string pattern = @"([\s;:,.()\[\]<>+\-\*/=])";

foreach (var line in lines)

{

lineIndex++;

string res = line.TrimStart();

string[] tokens = Regex.Split(res, pattern);

foreach (var token in tokens)

{

string descriptor = "";

if (analyzer.Keywords.Contains(token))

{

descriptor = "(K1" + "," + (analyzer.Keywords.IndexOf(token) + 1) + ")";

}

else if (analyzer.Identifiers.Contains(token))

{

descriptor = "(K3" + "," + (analyzer.Identifiers.IndexOf(token) + 1) + ")";

}

else if (analyzer.Constants.Contains(token))

{

descriptor = "(K4" + "," + (analyzer.Constants.IndexOf(token) + 1) + ")";

}

else if (analyzer.Delimiters.Contains(token))

{

descriptor = "(K2" + "," + (analyzer.Delimiters.IndexOf(token) + 1) + ")";

}

descriptorText += descriptor;

}

descriptorText += "\n";

}

return descriptorText;

}

static void Main(string[] args)

{

string code = @"

program MyFirstProgram;

var

y: integer;

x: integer;

begin

y = 2;

x = sin(y);

end.";

LexicalAnalyzer analyzer = new LexicalAnalyzer();

analyzer.Analyze(code);

Console.WriteLine("Keywords:");

foreach (var keyword in analyzer.Keywords)

{

Console.WriteLine(keyword);

}

Console.WriteLine("\nDelimiters:");

foreach (var delimiter in analyzer.Delimiters)

{

Console.WriteLine(delimiter);

}

Console.WriteLine("\nIdentifiers:");

foreach (var identifier in analyzer.Identifiers)

{

Console.WriteLine(identifier);

}

Console.WriteLine("\nConstants:");

foreach (var constant in analyzer.Constants)

{

Console.WriteLine(constant);

}

Console.WriteLine();

Console.WriteLine("Дескриптор:");

Console.WriteLine(GenerateDescriptor(code, analyzer));

}

}

class LexicalAnalyzer

{

public List<string> Keywords { get; private set; }

public List<string> Delimiters { get; private set; }

public List<string> Identifiers { get; private set; }

public List<string> Constants { get; private set; }

public LexicalAnalyzer()

{

Keywords = new List<string>();

Delimiters = new List<string>();

Identifiers = new List<string>();

Constants = new List<string>();

}

public void Analyze(string code)

{

string[] keywords = { "program", "var", "begin", "end", "if", "then", "else", "for", "to", "do", "sin", "cos", "tan", "exp", "integer", "real", "const", "array", "of", "Writeln" };

string[] delimiters = { ";", ":", ",", ".", "(", ")", "[", "]", "<", ">", "+", "-", "\*", "/", "=" };

string[] tokens = code.Split(new char[] { ';', ':', ',', '.', '(', ')', '[', ']', '<', '>', '+', '-', '\*', '/', '=', ' ', '\n', '\t', '\r' }, StringSplitOptions.RemoveEmptyEntries);

string pattern = @"[^;:,.()\[\]<>+\-\*/=]+";

string result = Regex.Replace(code, pattern, " ");

string[] razd = result.Split(' ');

foreach (string token in tokens)

{

// Игнорирует пустые токены

if (string.IsNullOrWhiteSpace(token))

{

continue;

}

// Если токен это кейворд

if (keywords.Contains(token))

{

AddToTable(token, Keywords);

}

// Если токен это идентификатор

else if (IsIdentifier(token))

{

AddToTable(token, Identifiers);

}

// Если токен это Константа

else if (IsConstant(token))

{

AddToTable(token, Constants);

}

}

foreach (string token in razd)

{

if (token.Length > 1)

{

string sub = token.Substring(0, 1);

if (delimiters.Contains(sub))

{

AddToTable(sub, Delimiters);

}

}

// CЕсли токен это делимитер

if (delimiters.Contains(token))

{

AddToTable(token, Delimiters);

}

}

}

private void AddToTable(string token, List<string> table)

{

if (!table.Contains(token))

{

table.Add(token);

}

}

private bool IsIdentifier(string token)

{

return Regex.IsMatch(token, @"^[a-zA-Z\_]\w\*$");

}

private bool IsConstant(string token)

{

return Regex.IsMatch(token, @"^\d+(&#46;\d+)?$");

}

}

}

**6.Отладочный пример работы программы**

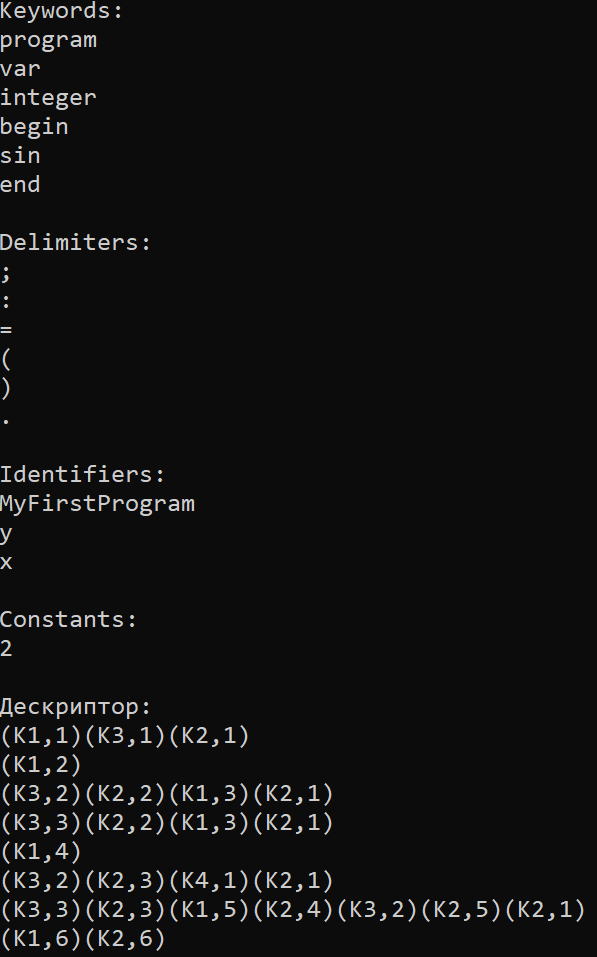


Рисунок 3.Пример работы программы

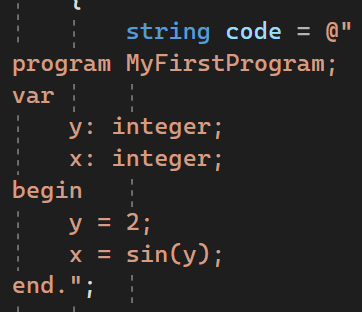


Рисунок 4.Входные данные