|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра прикладной математики | | | |
| Лабораторная работа № 2 | | | |
| по дисциплине «Языки программирования и методы трансляции» | | | |
| **Разработка и реализация блока лексического анализа (Сканер)** | | | |
|  | Группа | ПМ-84 |
| Бригада | 6 |
| Вариант | 6 |
| Студенты: | Ефименко Александра |
|  | Каральчук артем |
|  | кутузова ирина |
| Преподаватель | еланцева ирина леонидовна |
| Дата | 24.03.2021 |
|  | | | |
| Новосибирск | | | |

1. **Цели и задачи проекта**

**Цель:**

Изучить методы лексического анализа. Получить представление о методах обработки лексических ошибок. Научиться проектировать сканер на основе детерминированных конечных автоматов.

**Задачи:**

В соответствии с выбранным вариантом задания к лабораторным работам разработать и реализовать лексический анализатор на основе детерминированных конечных автоматов. Исходными данными для сканера является программа на языке С++ и постоянные таблицы, реализованные в лабораторной работе №1. Результатом работы сканера является создание файла токенов, переменных таблиц (таблицы символов и таблицы констант) и файла сообщений об ошибках.

1. **Вид, структура входных и выходных данных**

Входные данные –постоянные таблицы, исходная программа на языке С++

Постоянные таблицы формируются также, как и в 1 лабораторной работе.

Исходная программа – файл txt

Выходные данные – переменные таблицы, файл токенов

Переменные таблицы формируются по ходу заполнения токенов и хранятся как VariableTable

Файл токенов:

Токен - ( номер таблицы, номер строки )

Каждая строка файла токенов соответствует строке исходной программы

1. **Построение детерминированного конечного автомата**
2. **Алгоритм**
3. **Тексты программ**

**Program.cs:**

using System;

namespace Lab\_1

{

class Program

{

static void Main()

{

ConstantTable CTOperators = new ConstantTable($"C:\\Users\\pm82k\\source\\repos\\LoPaMoT\\LoPaMoT\\Operators.txt");

ConstantTable CTDivisions = new ConstantTable($"C:\\Users\\pm82k\\source\\repos\\LoPaMoT\\LoPaMoT\\Divisions.txt");

ConstantTable CTKeyWoeds = new ConstantTable($"C:\\Users\\pm82k\\source\\repos\\LoPaMoT\\LoPaMoT\\KeyWords.txt");

VariableTable VTConstants = new VariableTable();

VariableTable VTIdentificators = new VariableTable();

Scanner scanner = new Scanner(CTOperators, CTDivisions, CTKeyWoeds, VTConstants, VTIdentificators);

scanner.LexicalAnalysis($"C:\\Users\\pm82k\\source\\repos\\LoPaMoT\\LoPaMoT\\program.txt");

}

}

}

**LexicalAnalysis.cs:**

using System;

using System.Collections.Generic;

using System.IO;

namespace Lab\_1

{

class Scanner

{

public ConstantTable Operations;

public ConstantTable Divisions;

public ConstantTable KeyWords;

StreamWriter f;

VariableTable Constants;

VariableTable Identificators;

bool flagBlockComment = false;

string comment = "";

public Scanner(ConstantTable Operations, ConstantTable Divisions, ConstantTable KeyWords,

VariableTable Constants, VariableTable Identificators)

{

this.Operations = Operations;

this.Divisions = Divisions;

this.KeyWords = KeyWords;

this.Constants = Constants;

this.Identificators = Identificators;

}

enum Statuses : int

{

Start = 0, ReadName, ReadNum, ReadChar, ReadComment, ReadOperator, End, Error

}

public void ReadName()

{

}

public List<int[]> FiniteStateMachine(string line)

{

Statuses Status = Statuses.Start;

char[] letters = line.ToCharArray();

string name = "";

bool flag\_constant = false;

List<int[]> Tokens = new List<int[]>();

char openchar = ' ';

bool wasPoint = false;

bool flag\_comment = false;

bool flagLineComment = false;

foreach (char letter in letters)

{

if (Status == Statuses.End && letter != '\n')

{

Status = Statuses.Start;

}

if (Status == Statuses.ReadChar)

{

if ("\'\"".Contains(letter) && name != "\\")

{

if (name.Length == 0 && letter == '\'')

{

Status = Statuses.Error;

}

else

if (letter == openchar && (name.Length <= 1 || name.Length == 2 && name[0] == '\\'))

Status = Statuses.End;

else

Status = Statuses.Error;

}

else

{

name += letter;

}

}

if (Status == Statuses.Start)

{

if (flagBlockComment)

{

Status = Statuses.ReadComment;

}

else

{

openchar = ' ';

name = "";

flag\_constant = false;

wasPoint = false;

flag\_comment = false;

flagLineComment = false;

if ("qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM\_".Contains(letter))

{

Status = Statuses.ReadName;

}

else if ("1234567890.".Contains(letter))

{

flag\_constant = true;

Status = Statuses.ReadNum;

}

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

{

flag\_constant = true;

Status = Statuses.ReadChar;

openchar = letter;

}

else if (letter == ';' || letter == ',')

{

name = letter.ToString();

Status = Statuses.End;

}

else if (Operations.SearchIsExist(letter.ToString()) || letter == '!')

{

Status = Statuses.ReadOperator;

}

else if (letter == '/')

{

Status = Statuses.ReadComment;

}

else if (letter == '\n')

{

Status = Statuses.End;

}

}

}

if (Status == Statuses.ReadName)

{

if (letter != ';' && letter != ' ' && letter != ',' && letter != '\n')

{

if ("qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM\_".Contains(letter) || "1234567890".Contains(letter))

name += letter;

else

Status = Statuses.Error;

}

else

Status = Statuses.End;

}

if (Status == Statuses.ReadNum)

{

if ("1234567890.".Contains(letter))

{

if (name == "0" && letter != '.')

name = "";

Status = Statuses.ReadNum;

name += letter;

if (letter == '.')

if (wasPoint)

Status = Statuses.Error;

else

wasPoint = true;

}

else if (letter == ';' || letter == ' ' || letter == '\n')

{

if (name == ".")

Status = Statuses.Error;

else

Status = Statuses.End;

}

else

{

Status = Statuses.Error;

}

}

if (Status == Statuses.ReadComment)

{

if (comment == "/")

{

if (letter == '/')

{

Console.WriteLine("линейный коммент");

flagLineComment = true;

}

else if (letter == '\*')

{

Console.WriteLine("блочный коммент");

flagBlockComment = true;

}

else

{

Status = Statuses.Error;

}

}

else if (flagLineComment && letter == '\n')

{

Status = Statuses.End;

flagLineComment = false;

comment = "";

}

else if (flagBlockComment && comment[^2] == '\*' && comment[^1] == '/' && comment.Length > 3)

{

flagBlockComment = false;

Status = Statuses.Start;

comment = "";

}

if (letter == '\n')

Status = Statuses.End;

else

comment += letter;

}

if (Status == Statuses.ReadOperator)

{

name += letter;

if (name[0] != '!' && name.Length == 1)

Status = Statuses.End;

else

if (name.Length == 2)

if (name == "!=")

Status = Statuses.End;

else

Status = Statuses.Error;

}

if (Status == Statuses.End)

{

if (!flagBlockComment && !flagLineComment)

{

if (letter != '\n')

{

int[] token = GetToken(name, flag\_constant);

if (token[1] != -1)

{

Tokens.Add(token);

if ((letter == ';' || letter == ',') && name != letter.ToString())

Tokens.Add(GetToken(letter.ToString(), flag\_constant));

}

}

}

}

}

if (Status != Statuses.End)

{

Console.WriteLine("ERROR");

comment = "";

}

return Tokens;

}

private int[] GetToken(string name, bool flag)

{

int id = -1;

if (name.Length <= 2)

{

if ((id = Operations.SearchIdByName(name)) != -1) return new int[] { 0, id };

if ((id = Divisions.SearchIdByName(name)) != -1) return new int[] { 1, id };

}

if ((id = KeyWords.SearchIdByName(name)) != -1) return new int[] { 2, id };

if (flag)

{

id = Constants.SearchIdByName(name); return new int[] { 3, id };

}

else

{

if (name.Length != 0)

id = Identificators.SearchIdByName(name); return new int[] { 4, id };

}

}

public void LexicalAnalysis(string filename)

{

using (f = new StreamWriter("Tokenы.txt")){

string[] chains = File.ReadAllLines(filename);

if (chains.Length == 0)

{

Console.WriteLine("ERRoR");

}

foreach (var line in chains)

{

WriteTokens(FiniteStateMachine(line + " \n"));

}

}

}

private void WriteTokens(List<int[]> Tokens)

{

foreach (var token in Tokens)

{

f.Write("( " + token[0] + ", " + token[1] + " ) ");

Console.Write("( " + token[0] + ", " + token[1] + " ) ");

}

f.Write("\n");

Console.Write('\n');

}

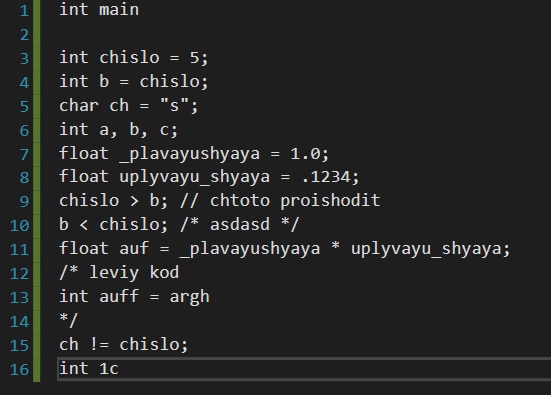
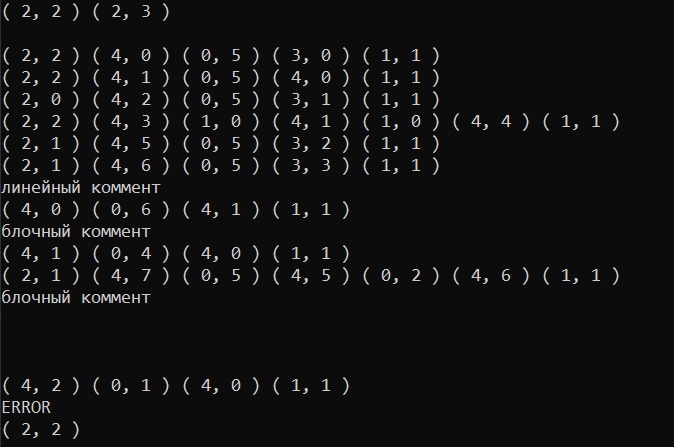
}

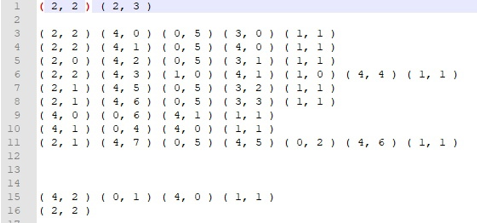
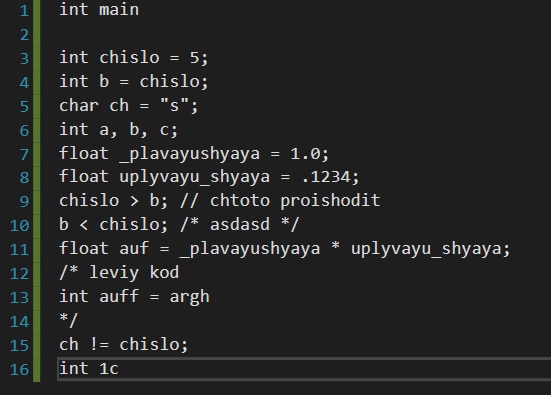
}

1. **Тестовые примеры**

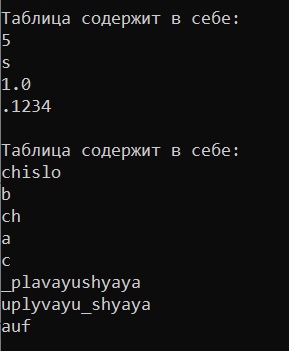
Для наглядности тестирования, ошибки и комментарии были выписаны, таким образом для строки они выводятся перед токенами которые были найдены в строке.

1. Тест на успешную обработку кода

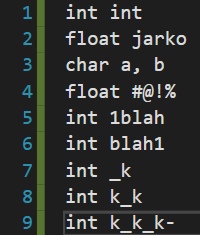
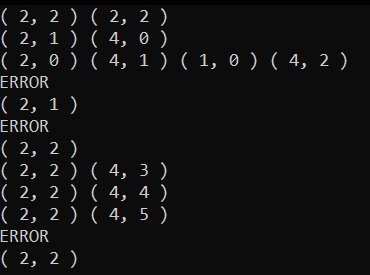
 



Переменные таблицы:

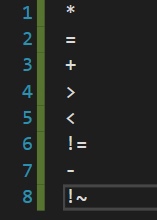
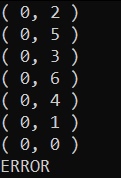


1. Тест на обработку идентификаторов + ключевых слов

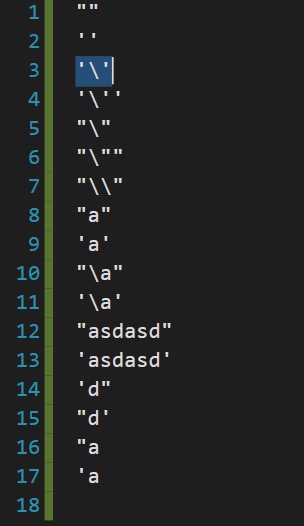
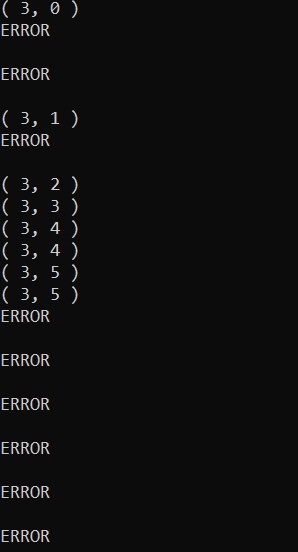
 

Переменные таблицы:

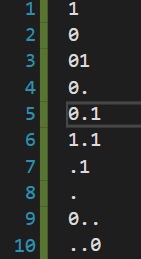
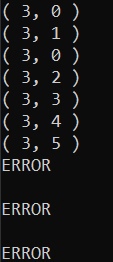
1. Тест на обработку операторов

1. Тест на обработку символьных констант

1. Тест на обработку чисел

1. Обработка комментариев

