ГУАП

КАФЕДРА 44

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, канд. техн. наук |  |  |  | Н. В. Кучин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| ПОСТРОЕНИЕ РАСПОЗНОВАТЕЛЯ ДЛЯ РЕГУЛЯРНОЙ ГРАММАТИКИ И ПОСТРОЕНИЕ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА |
| по курсу: СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4941 |  |  |  | Н.С. Горбунов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

1. **Цель работы:** изучение основных понятий теории регулярных языков и грамматик, ознакомление с назначением и принципами работы конечных автоматов (КА), получение практических навыков построения КА на основе заданной регулярной грамматики; ознакомление с назначением и принципами работы лексических анализаторов (сканеров), получение практических навыков построения сканера на примере заданного простейшего входного языка.
2. **Вариант 7**

Входной язык содержит арифметические выражения, разделенные символом ; (точка с запятой). Арифметические выражения состоят из идентификаторов, римских чисел, знака присваивания (:=), знаков операций +, -, \*, / и круглых скобок.

1. **Возможные входные данные языка:**

* Буквы латинского алфавита составляющие идентификаторы и римские цифры
* Специальные символы для обозначения присваивания и конца

строки - «:=» и «;» соответственно

1. **Граф состояний**

Для удобства были обозначены следующие множества:

локальное множество символов, неподходящий для данного конкретного состояния

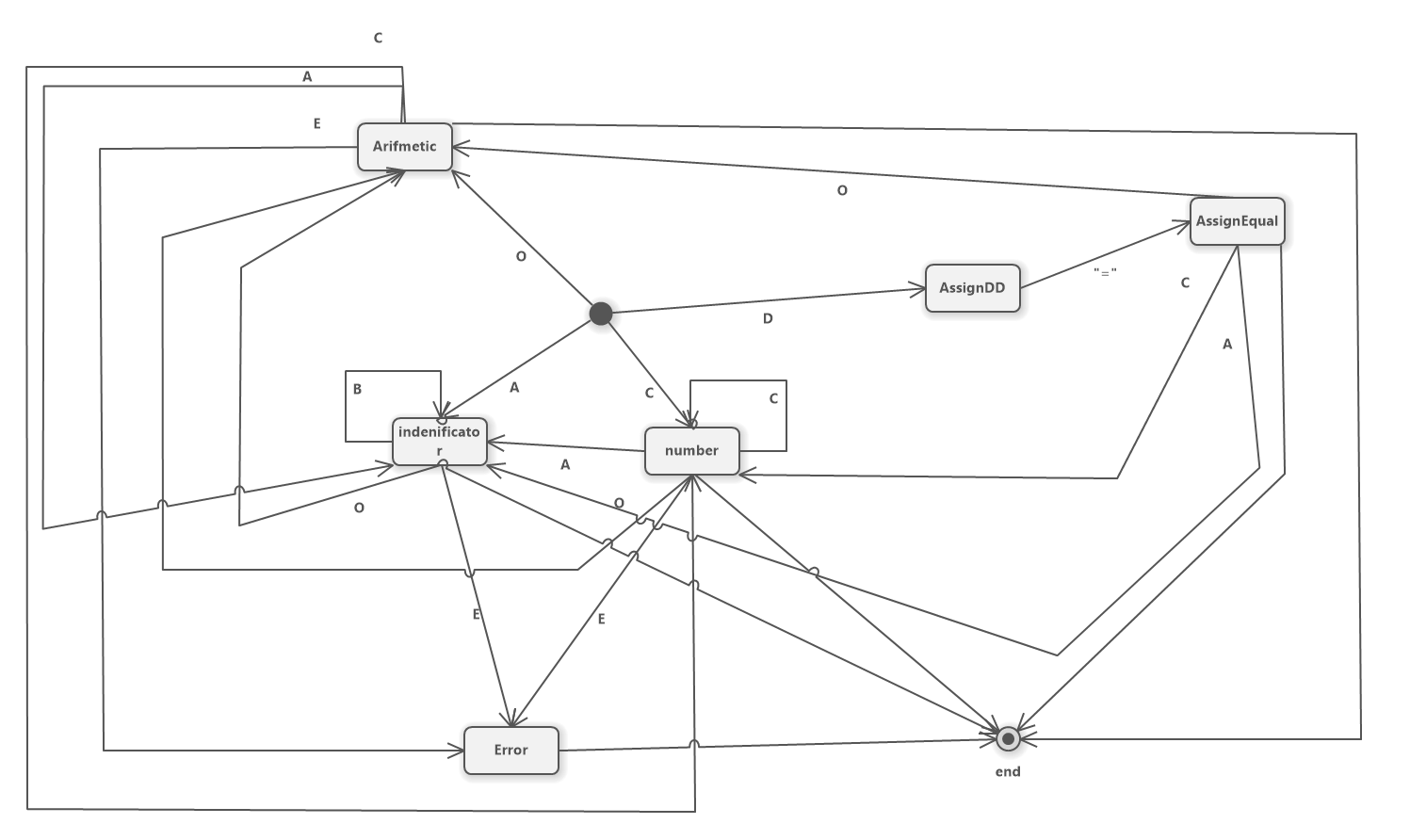
****

Рисунок 1. Граф состояний

1. **Скриншот работы программы**

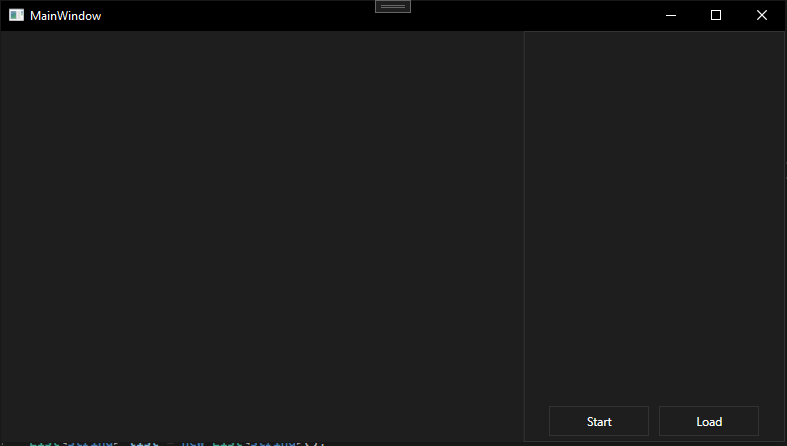


Рисунок 2. Пустое окно

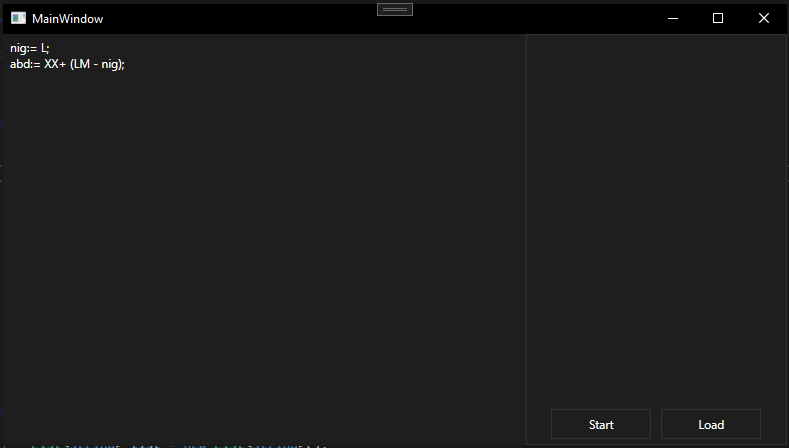


Рисунок 3. Введенные данные

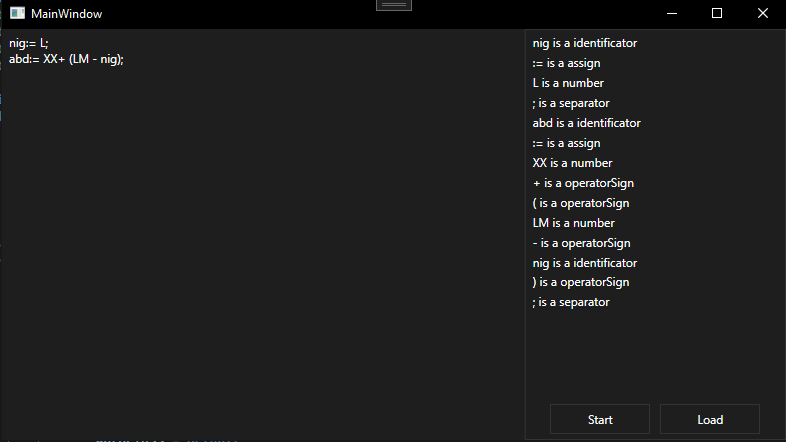


Рисунок 4. Результат работы программы

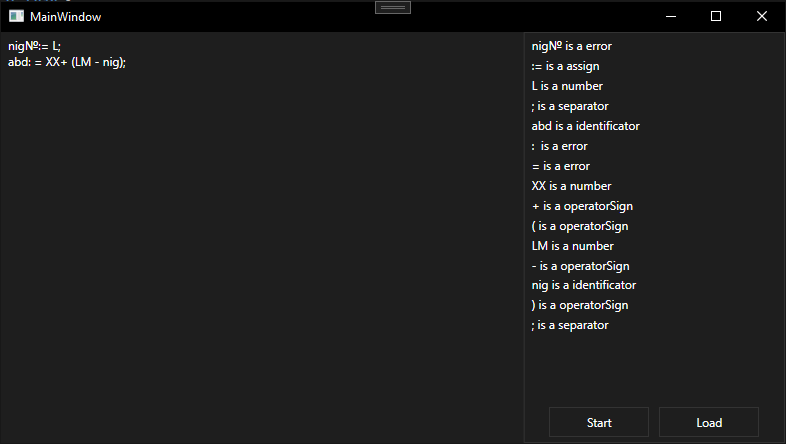


Рисунок 5. Результат работы программы с ошибкой

1. Программа

Был реализован класс, выполняющий лексический анализ выражения по передаваемой строке.

Листинг 1

internal class LingAnalizer

{

string input;

public LingAnalizer(string text)

{

if (text is null || input == "")

throw new NullReferenceException() ;

input = text.Replace('\r', ' ').Replace('\n', ' ');// удаление переносов строк

}

// описание множеств

static string B = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

static string C = "IVXLCDM";

static string A = "abcdefghijklmnopqrstuvwxyzABEFGHJKNOPQRSTUWYZ";

static char S = ';';

static char P = ' ';

static char D = ':';

static char Q = '=';

static string O = "+-\*/()";

string word;

public List<string> Solve()

{

List<string> list = new List<string>();

State state = 0; // текущее состояние

word = ""; //собираемое слово

WordClass wordClass = default;

foreach (char s in input) //посимвольный перебор

{

switch(state) //переход в зависимости от состояния

/\* каждое состояние выполняет проверку нового символа (конструкция if)

\* и выполняет логику этого состояния

\*/

{

#region Init

// начальное состояние. Первичное распеделение

case State.init:

word = "";

state = State.error;

wordClass = default;

if (A.Contains(s)) state = State.identificator;

if (C.Contains(s)) state = State.numbers;

if (D == s) state = State.assignDD;

if (O.Contains(s)) state = State.arifmetic;

if (s == S) state = State.separator;

if (s == P) state = State.init;

break;

#endregion

#region Identificator

// допускаем, что это индентификатор. Если будет введен неверный символ, то всё слово - ошибка

case State.identificator:

wordClass = WordClass.identificator;

state = State.error;

if (B.Contains(s))

{

state = State.identificator;

}

if (O.Contains(s))

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.arifmetic;

}

if (s == P)

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.init;

}

if (s == S)

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.separator;

}

if (s == D)

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.assignDD;

}

break;

#endregion

#region Number

// состояние римской цифры. Может стать идентификатором, если будет лишняя буква

case State.numbers:

wordClass = WordClass.number;

state = State.error;

if (A.Contains(s)) state = State.identificator;

if (C.Contains(s)) state = State.numbers;

if (O.Contains(s))

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.arifmetic;

}

if (s == P)

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.init;

}

if (s == S)

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.separator;

}

if (s == D)

{

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.assignDD;

}

break;

#endregion

#region Arifmetic

// арифметические знаки. Они все односимовольные, а значит, что каждый символ будет определен как отдельеный арифметический знак

case State.arifmetic:

wordClass = WordClass.operatorSign;

state = State.error;

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

if (s == P) state = State.init;

if (s == S) state = State.separator;

if (s == D) state = State.assignDD;

if (A.Contains(s)) state = State.identificator;

if (C.Contains(s)) state = State.numbers;

break;

#endregion

#region AssignDD

/\* промежуточное состояние для знака присвоения

\* Так как символ ":" не используется больше нигде,

\* то если следущий символ не "=", тогда будет ошибка

\*/

case State.assignDD:

wordClass = WordClass.assign;

state = State.error;

if (s == Q) state = State.assignEqual;

break;

#endregion

#region AssignEqual

// заключающее состояние знака присвоения

case State.assignEqual:

wordClass = WordClass.assign;

state = State.error;

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

if (A.Contains(s)) state = State.identificator;

if (C.Contains(s)) state = State.numbers;

if (O.Contains(s)) state = State.arifmetic;

if (s == P) state = State.init;

if (s == S) state = State.separator;

break;

#endregion

#region Separator

/\*состояние разделителя. В теории, можно было бы обойтись и без него,

\* но так получается детерминированный автомат на switch-case :)

\*/

case State.separator:

wordClass = WordClass.separator;

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.error;

wordClass = default;

if (A.Contains(s)) state = State.identificator;

if (C.Contains(s)) state = State.numbers;

if (D == s) state = State.assignDD;

if (O.Contains(s)) state = State.arifmetic;

if (s == S) state = State.separator;

if (s == P) state = State.init;

break;

#endregion

#region Error

//состояние ошибки. Просто выводит слово как ошибочное

case State.error:

wordClass = WordClass.error;

list.Add(String.Format("{0} is a {1}", word, wordClass));

word = "";

state = State.error;

wordClass = default;

if (A.Contains(s)) state = State.identificator;

if (C.Contains(s)) state = State.numbers;

if (D == s) state = State.assignDD;

if (O.Contains(s)) state = State.arifmetic;

if (s == S) state = State.separator;

if (s == P) state = State.init;

break;

#endregion

}

word += s;

}

return list;

}

}

internal enum State // перечисление состояний

{

init,

numbers,

identificator,

assignDD,

assignEqual,

arifmetic,

error,

separator,

}

internal enum WordClass // перечисление типов лексем

{

error,

identificator,

assign,

operatorSign,

number,

separator

}

**Вывод:** Изучил основные понятия теории регулярных языков и грамматик, ознакомился с назначением и принципами работы конечных автоматов (КА), получил практические навыки построения КА на основе заданной регулярной грамматики; ознакомился с назначением и принципами работы лексических анализаторов (сканеров), получил практические навыки построения сканера на примере заданного простейшего входного языка.