**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА (САМАРСКИЙ УНИВЕРСИТЕТ)»**

ИНСТИТУТ ИНФОРМАТИКИ, МАТЕМАТИКИ И ЭЛЕКТРОНИКИ

Факультет информатики

Кафедра программных систем

Дисциплина

**Теория автоматов и формальных языков**

**ОТЧЕТ**

по лабораторной работе

**«Синтаксический анализ автоматных языков.»**

Вариант № 17

Студент: Гижевская В.Д.

Группа: 6213-020302D

Преподаватель: Литвинов В.Г.

Оценка:

Дата: 13.12.2019

Самара 2019**СОДЕРЖАНИЕ**

[Лабораторная работа «Синтаксический анализ автоматных языков» 3](#_Toc26748246)

[1. Задание 3](#_Toc26748247)

[2. Граф состояний 4](#_Toc26748248)

[3. Листинг программы 6](#_Toc26748249)

[4. Результаты тестирования 17](#_Toc26748250)

[Список использованных источников 18](#_Toc26748251)

Лабораторная работа   
«Синтаксический анализ автоматных языков»

## Задание

**Синтаксис:** Написать программу синтаксического анализа оператора заголовка цикла языка Modula-2 (язык регистро-независимый). Грамматика имеет следующий вид:

|  |  |  |  |
| --- | --- | --- | --- |
| № |  |  | Правила |
| (1) | *start* | = **FOR** *var* **:=** *int-numb* **TO** *int-numb* [ **BY** *int-numb* ] **DO** | |
| (2) | *var* | = | *id* [**[** *indexes* **]**] |
| (3) | *indexes* | = *index* { **,** *index* } | |
| (4) | *index* | = | *id* | *int-numb* |

* *id* — идентификатор языка Modula-2, начинается с буквы или знака подчеркивания, далеемогут следовать буквы, цифры и знаки подчеркивания.

Ограничения:

**–** имеет длину не более 8 символов;

**–** не является зарезервированным словом: **FOR** , **TO** , **BY** , **DO** .

* *int-numb* — целое число в диапазоне -32768 – 32767.

**Семантика:**

* Построить и вывести на печать по завершении анализа таблицы идентификаторов и констант.
* Учесть перечисленные ограничения на идентификаторы и константы.
* Определить, сколько раз выполняется цикл.
* Сообщать об ошибках при анализе, указывая курсором место ошибки и ее содержание.

**Примеры правильных цепочек:**

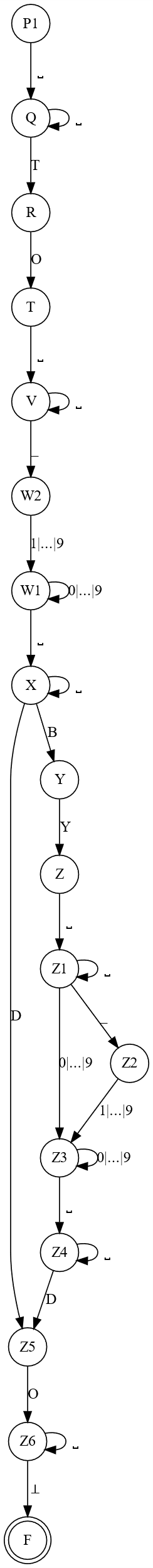
**FOR** I := 10 **TO** 20 **BY** 2 **DO**

**FOR** ABC1D [I,25, J,75]:= -100 **TO** -120 **BY** 5 **DO**

**FOR** A3:= 1 **TO** 200

## Граф состояний

Рисунок 2.1 – Граф состояний конечного автомата (начало)

Рисунок 2.1 – Граф состояний конечного автомата (окончание)

## Листинг программы

using System;

using System.Collections.Generic;

namespace Analyzer

{

public class Analyzer

{

private readonly static string[] RESERVED = { "for", "to", "by", "do" };

// Позиция курсора

private int position;

// Входная цепочка

private string chain;

// Текст ошибки

private string errorMessage;

// Позиция ошибки в строке

private int errorPosition;

// Список идентификаторов

private LinkedList<string> identifiers;

// Список констант

private LinkedList<string> constants;

// Количество итераций

private int iterations;

public Analyzer(string chain)

{

position = 0;

errorMessage = "";

errorPosition = -1;

iterations = 0;

identifiers = new LinkedList<string>();

constants = new LinkedList<string>();

this.chain = chain;

AnalyseChain();

}

public int ErrorPosition {

get

{

return errorPosition;

}

}

public string ErrorMessage

{

get

{

return errorMessage;

}

}

public LinkedList<string> Identifiers

{

get

{

return identifiers;

}

}

public LinkedList<string> Constants

{

get

{

return constants;

}

}

public int Iterations

{

get

{

return iterations;

}

}

// Метод анализа строки

private bool AnalyseChain()

{

State state = State.S;

string identifier = string.Empty;

string constant = string.Empty;

errorMessage = "";

short numFor = 0;

short numBy = 1;

short numTo = 0;

while (state != State.E && state != State.F)

{

if (position >= chain.Length)

{

errorMessage = "Непредвиденный конец строки";

state = State.E;

break;

}

char ch = char.ToLower(chain[position]);

switch (state)

{

case State.S:

{

if (ch == 'f')

state = State.A;

else if (ch != ' ')

{

errorMessage = "Ожидалось: \"F\"";

state = State.E;

}

break;

}

case State.A:

{

if (ch == 'o')

state = State.B;

else

{

errorMessage = "Ожидалось: \"O\"";

state = State.E;

}

break;

}

case State.B:

{

if (ch == 'r')

state = State.C;

else

{

errorMessage = "Ожидалось: \"R\"";

state = State.E;

}

break;

}

case State.C:

{

if (ch == ' ')

state = State.D;

else

{

errorMessage = "Ожидалось: пробел";

state = State.E;

}

break;

}

case State.D:

{

if (ch == '\_' || char.IsLetter(ch))

{

identifier = ch.ToString();

state = State.I1;

}

else if (ch != ' ')

{

errorMessage = "Ожидалось: символ подчеркивания, буква или пробел";

state = State.E;

}

break;

}

case State.I1:

{

if (ch == '\_' || char.IsLetterOrDigit(ch))

identifier += ch;

else

{

if (!CheckIdentifier(identifier))

{

errorMessage = "Некорректный идентификатор";

state = State.E;

break;

}

if (!identifiers.Contains(identifier))

identifiers.AddLast(identifier);

if (ch == ':')

state = State.N;

else if (ch == ' ')

state = State.H;

else if (ch == '[')

state = State.G;

else

{

errorMessage = "Ожидалось: \" \", цифра, буква, \"\_\", \"[\", \":\"";

state = State.E;

}

}

break;

}

case State.G:

{

if (ch == '-')

{

constant = ch.ToString();

state = State.K;

}

else if (char.IsDigit(ch))

{

constant = ch.ToString();

state = State.J;

}

else if (ch == '\_' || char.IsLetter(ch))

{

identifier = ch.ToString();

state = State.I2;

}

else if (ch != ' ')

{

errorMessage = "Ожидалась числовая константа или идентификатор";

state = State.E;

}

break;

}

case State.H:

{

if (ch == '[')

state = State.G;

else if (ch == ':')

state = State.N;

else if (ch != ' ')

{

errorMessage = "Ожидалось: пробел, двоеточие или \"[\"";

state = State.E;

}

break;

}

case State.I2:

{

if (ch == '\_' || char.IsLetterOrDigit(ch))

identifier += ch;

else

{

if (!CheckIdentifier(identifier))

{

errorMessage = "Некорректный идентификатор";

state = State.E;

break;

}

if (!identifiers.Contains(identifier))

identifiers.AddLast(identifier);

if (ch == ',')

state = State.G;

else if (ch == ' ')

state = State.L;

else if (ch == ']')

state = State.M;

else

{

errorMessage = "Ожидалось: \" \", цифра, буква, \"\_\", \"]\", \",\"";

state = State.E;

}

}

break;

}

case State.K:

{

if (char.IsDigit(ch) && ch != '0')

{

constant += ch;

state = State.J;

}

else

{

errorMessage = "Ожидалось: цифра 1..9";

state = State.E;

}

break;

}

case State.J:

{

if (char.IsDigit(ch))

constant += ch;

else

{

if (!CheckConstant(constant))

{

errorMessage = "Значение константы не входит в диапазон -32768..32767";

state = State.E;

break;

}

constants.AddLast(constant);

if (ch == ',')

state = State.G;

else if (ch == ' ')

state = State.L;

else if (ch == ']')

state = State.M;

else

{

errorMessage = "Ожидалось: пробел, цифра, запятая или \"]\"";

state = State.E;

}

}

break;

}

case State.L:

{

if (ch == ',')

state = State.G;

else if (ch == ']')

state = State.M;

else if (ch != ' ')

{

errorMessage = "Ожидалось: пробел, запятая или \"]\"";

state = State.E;

}

break;

}

case State.M:

{

if (ch == ':')

state = State.N;

else if (ch != ' ')

{

errorMessage = "Ожидался пробел или двоеточие";

state = State.E;

}

break;

}

case State.N:

{

if (ch == '=')

state = State.O;

else

{

errorMessage = "Ожидалось: символ \"=\"";

state = State.E;

}

break;

}

case State.O:

{

if (ch == '-')

{

constant = ch.ToString();

state = State.P2;

}

else if (char.IsDigit(ch))

{

constant = ch.ToString();

state = State.P1;

}

else if (ch != ' ')

{

errorMessage = "Ожидалось: пробел, цифра или знак \"-\"";

state = State.E;

}

break;

}

case State.P2:

{

if (char.IsDigit(ch) && ch != '0')

{

constant += ch;

state = State.P1;

}

else

{

errorMessage = "Ожидалось: цифра 1..9";

state = State.E;

}

break;

}

case State.P1:

{

if (char.IsDigit(ch))

constant += ch;

else

{

if (!CheckConstant(constant))

{

errorMessage = "Значение константы не входит в диапазон -32768..32767";

state = State.E;

break;

}

constants.AddLast(constant);

numFor = short.Parse(constant);

if (ch == ' ')

state = State.Q;

else

{

errorMessage = "Ожидалось: пробел или цифра";

state = State.E;

}

}

break;

}

case State.Q:

{

if (ch == 't')

state = State.R;

else if (ch != ' ')

{

errorMessage = "Ожидалось: \"T\"";

state = State.E;

}

break;

}

case State.R:

{

if (ch == 'o')

state = State.T;

else

{

errorMessage = "Ожидалось: \"O\"";

state = State.E;

}

break;

}

case State.T:

{

if (ch == ' ')

state = State.V;

else

{

errorMessage = "Ожидалось: пробел";

state = State.E;

}

break;

}

case State.V:

{

if (ch == '-')

{

constant = ch.ToString();

state = State.W2;

}

else if (char.IsDigit(ch))

{

constant = ch.ToString();

state = State.W1;

}

else if (ch != ' ')

{

errorMessage = "Ожидалось: пробел, цифра или знак \"-\"";

state = State.E;

}

break;

}

case State.W2:

{

if (char.IsDigit(ch) && ch != '0')

{

constant += ch;

state = State.W1;

}

else

{

errorMessage = "Ожидалось: цифра 1..9";

state = State.E;

}

break;

}

case State.W1:

{

if (char.IsDigit(ch))

constant += ch;

else

{

if (!CheckConstant(constant))

{

errorMessage = "Значение константы не входит в диапазон -32768..32767";

state = State.E;

break;

}

constants.AddLast(constant);

numTo = short.Parse(constant);

if (ch == ' ')

state = State.X;

else

{

errorMessage = "Ожидалось: пробел или цифра";

state = State.E;

}

}

break;

}

case State.X:

{

if (ch == 'd')

state = State.Z5;

else if (ch == 'b')

state = State.Y;

else if (ch != ' ')

{

errorMessage = "Ожидалось: \"DO\" или \"BY\"";

state = State.E;

}

break;

}

case State.Y:

{

if (ch == 'y')

state = State.Z;

else

{

errorMessage = "Ожидалось: \"Y\"";

state = State.E;

}

break;

}

case State.Z:

{

if (ch == ' ')

state = State.Z1;

else

{

errorMessage = "Ожидалось: пробел";

state = State.E;

}

break;

}

case State.Z1:

{

if (ch == '-')

{

constant = ch.ToString();

state = State.Z2;

}

else if (char.IsDigit(ch))

{

constant = ch.ToString();

state = State.Z3;

}

else if (ch != ' ')

{

errorMessage = "Ожидалось: пробел, цифра или знак \"-\"";

state = State.E;

}

break;

}

case State.Z3:

{

if (char.IsDigit(ch))

constant += ch;

else

{

if (!CheckConstant(constant))

{

errorMessage = "Значение константы не входит в диапазон -32768..32767";

state = State.E;

break;

}

constants.AddLast(constant);

numBy = short.Parse(constant);

if (ch == ' ')

state = State.Z4;

else

{

errorMessage = "Ожидалось: пробел или цифра";

state = State.E;

}

}

break;

}

case State.Z2:

{

if (char.IsDigit(ch) && ch != '0')

{

constant += ch;

state = State.Z3;

}

else

{

errorMessage = "Ожидалось: цифра 1..9";

state = State.E;

}

break;

}

case State.Z4:

{

if (ch == 'd')

state = State.Z5;

else if (ch != ' ')

{

errorMessage = "Ожидалось: \"D\"";

state = State.E;

}

break;

}

case State.Z5:

{

if (ch == 'o')

state = position < chain.Length - 1 ?

State.Z6 :

State.F;

else

{

errorMessage = "Ожидалось: \"O\"";

state = State.E;

}

break;

}

case State.Z6:

{

if (ch == ' ' && position == chain.Length - 1)

state = State.F;

else if (ch != ' ')

{

errorMessage = "Ожидалось: пробел или конец входной строки";

state = State.E;

}

break;

}

default:

{

errorMessage = "Неизвестная ошибка";

state = State.E;

break;

}

}

++position;

}

if (state == State.E)

{

errorPosition = position - 1;

}

else

{

iterations = ((numTo - numFor) / numBy) + 1;

if (iterations < 0) iterations = 0;

}

return state == State.F;

}

private static bool CheckIdentifier(string identifier)

{

return identifier.Length <= 8 && Array.IndexOf(RESERVED, identifier) == -1;

}

private static bool CheckConstant(string constant)

{

return short.TryParse(constant, out \_);

}

// Перечисление возможных состояний автомата

private enum State

{

S, E, F,

A, B, C, D, I1, G, H, I2, K, J, L, M, N, O, P2, P1, Q,

R, T, V, W2, W1, X, Y, Z, Z1, Z3, Z2, Z4, Z5, Z6

}

}

}

## Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 4.1 – 4.2.

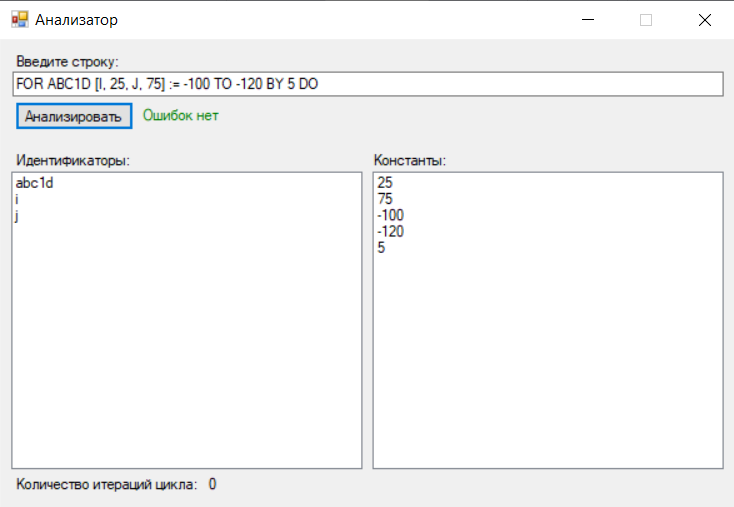


Рисунок 4.1 – Результат работы программы при вводе правильной цепочки

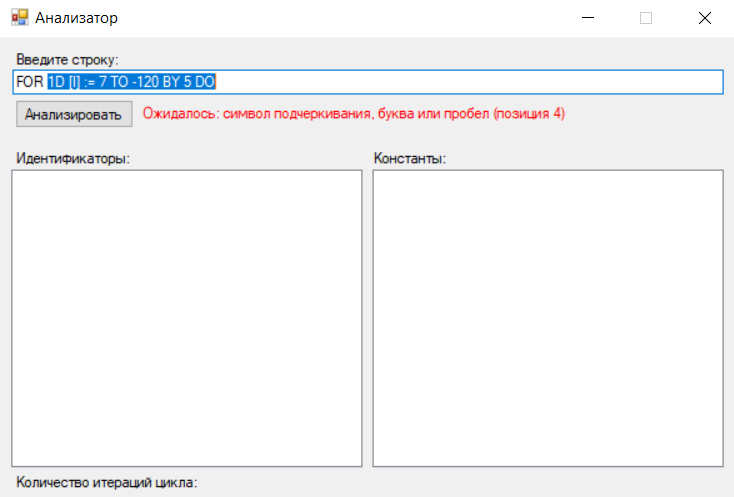


Рисунок 4.2 – Результат работы программы при вводе ошибочной цепочки

Список использованных источников

1. Е.И. Чигарина, М.А. Шамашов. Теория конечных автоматов и формальных языков. – Самара: изд-во СГАУ, 2007.
2. Е.И. Чигарина, В.Г. Литвинов. Теория формальных языков и грамматик. Методические указания. – Самара: изд-во Самарский университет, 2018.