МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА»

(САМАРСКИЙ УНИВЕРСИТЕТ)

Институт информатики и кибернетики

Кафедра информационных систем и технологий

ОТЧЕТ

по лабораторной работе по курсу

«Теория автоматов и формальных языков»

На тему:

**Синтаксический анализ автоматных языков**

Вариант № 9

Обучающийся в группе 6203-020302 М.А.Мананников

Руководитель Е.И.Чигарина

Самара 2022

ЗАДАНИЕ

Написать программу синтаксического анализа автоматного языка операторов присоединения языка Turbo Pascal, имеющего вид:

**WITH** <переменная> **DO** <оператор присваивания>**;**

<переменная> :: = <идентификатор>[**,**<идентификатор>]<идентификатор>**.**<иден-тификатор>

<оператор присваивания> :: = <левая часть> **:=** <правая часть>

⎡⎧ **[<**идентификатор**>]**⎫⎤

<левая часть> :: = <идентификатор> ⎢⎨ ⎬⎥

⎣⎩**[<**целая константа**>]**⎭⎦

<правая часть> :: = <оператор>[<операция><оператор>]

⎡⎧ **[**<идентификатор>**]**⎫⎤

<оператор> :: = <идентификатор> ⎢⎨ ⎬⎥<константа

⎣⎩**[**<константа целая>**]**⎭⎦ любая>

⎧ + ⎫

⎪ - ⎪

<операция> :: = ⎨ / ⎬

⎪ \* ⎪

⎪ div ⎪

⎩ mod ⎭

<идентификатор> - идентификатор языка Turbo Pascal, начинается с буквы или знака подчеркивания, включает буквы, цифры, не допускает пробелы и специальные символы, ввести ограничение на длину (не более 8 символов) и не может быть зарезервированным словом (WITH, DO, DIV, MOD);

<константа целая> - целое число в диапазоне -32768 ÷ +32767.

<константа любая> - целое число, число с фиксированной точкой, число с плавающей точкой.

Семантика:

Построить таблицу идентификаторов и констант. Учесть перечисленные выше ограничения на идентификаторы и константы.

Сообщать об ошибках при анализе цепочек языка, указывая курсором место ошибки и ее содержание.

Примеры правильных цепочек:

WITH A DO B := 13.3E - 10;

WITH A, B DO C := ‘текст’;

WITH A.D DO L := 18.1 + D;

WITH A.D, C.M DO \_A\_N [-3] := 3E + 2 - \_D1 [ I ];

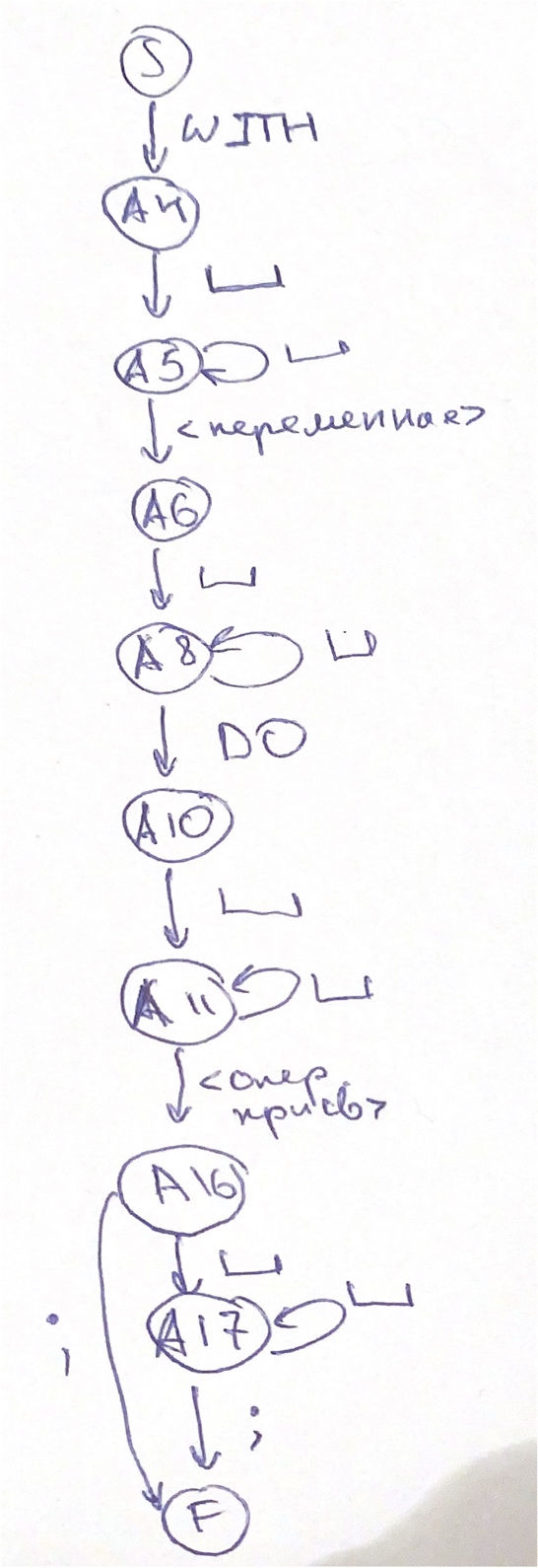
ГРАФЫ СОСТОЯНИЙ АНАЛИЗАТОPА

Рисунок 1. Граф состояния анализатора оператора WITH

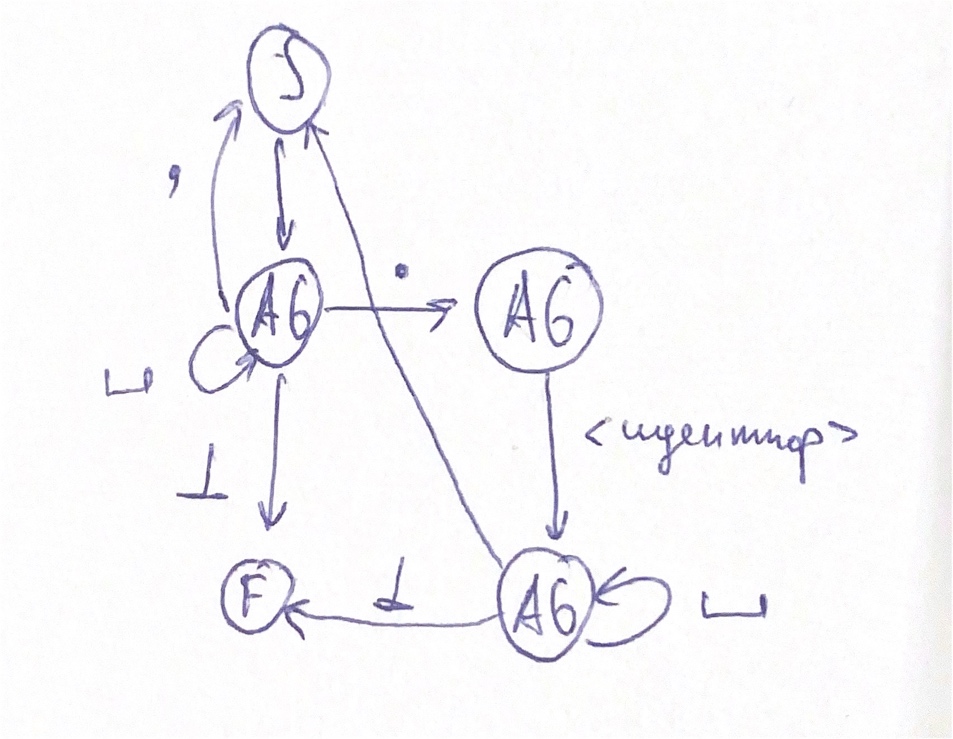


Рисунок 2. Граф состояния анализатора <переменная>

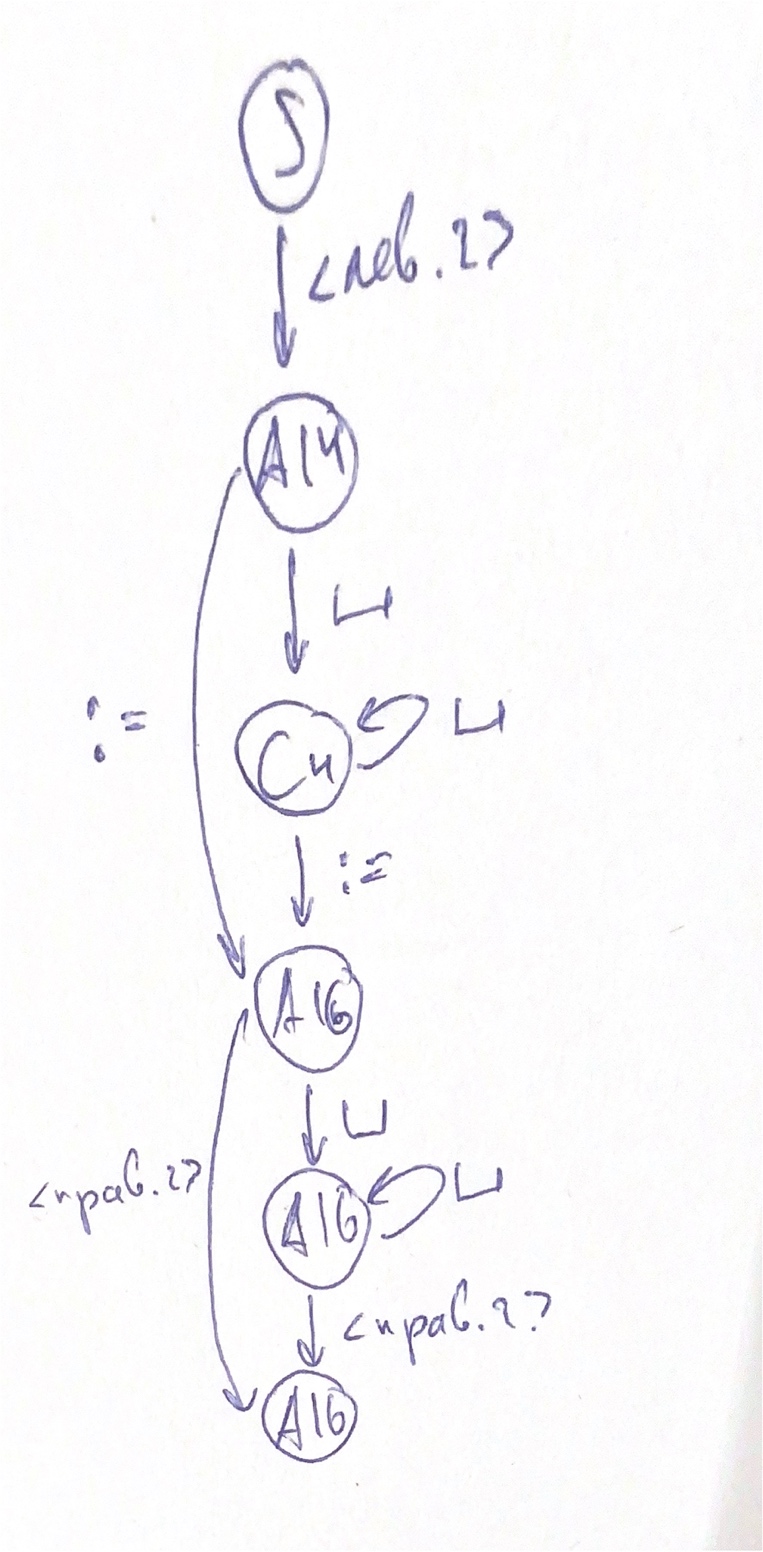


Рисунок 3. Граф состояния анализатора <оператор присваивания>

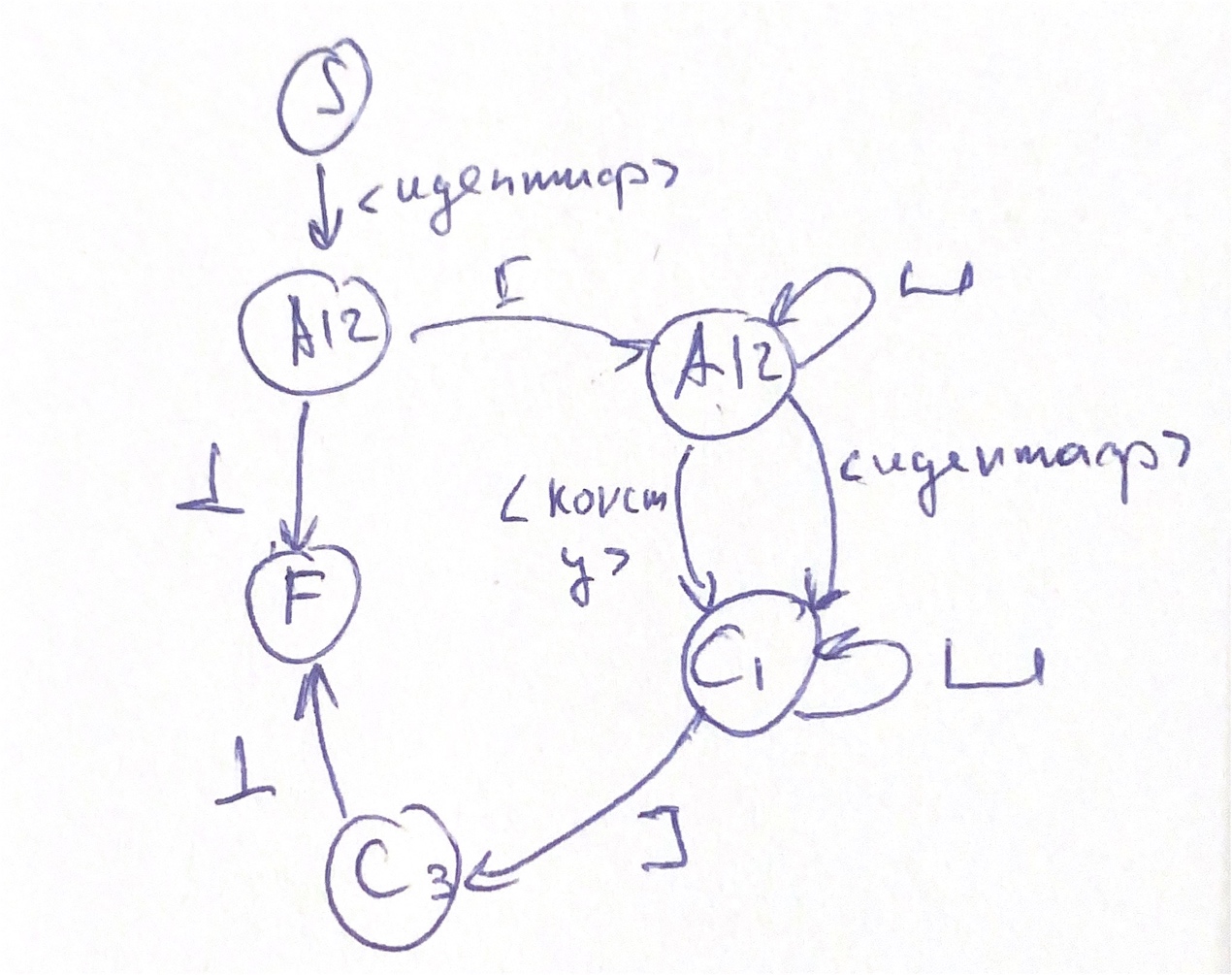


Рисунок 4. Граф состояния анализатора <левая часть>

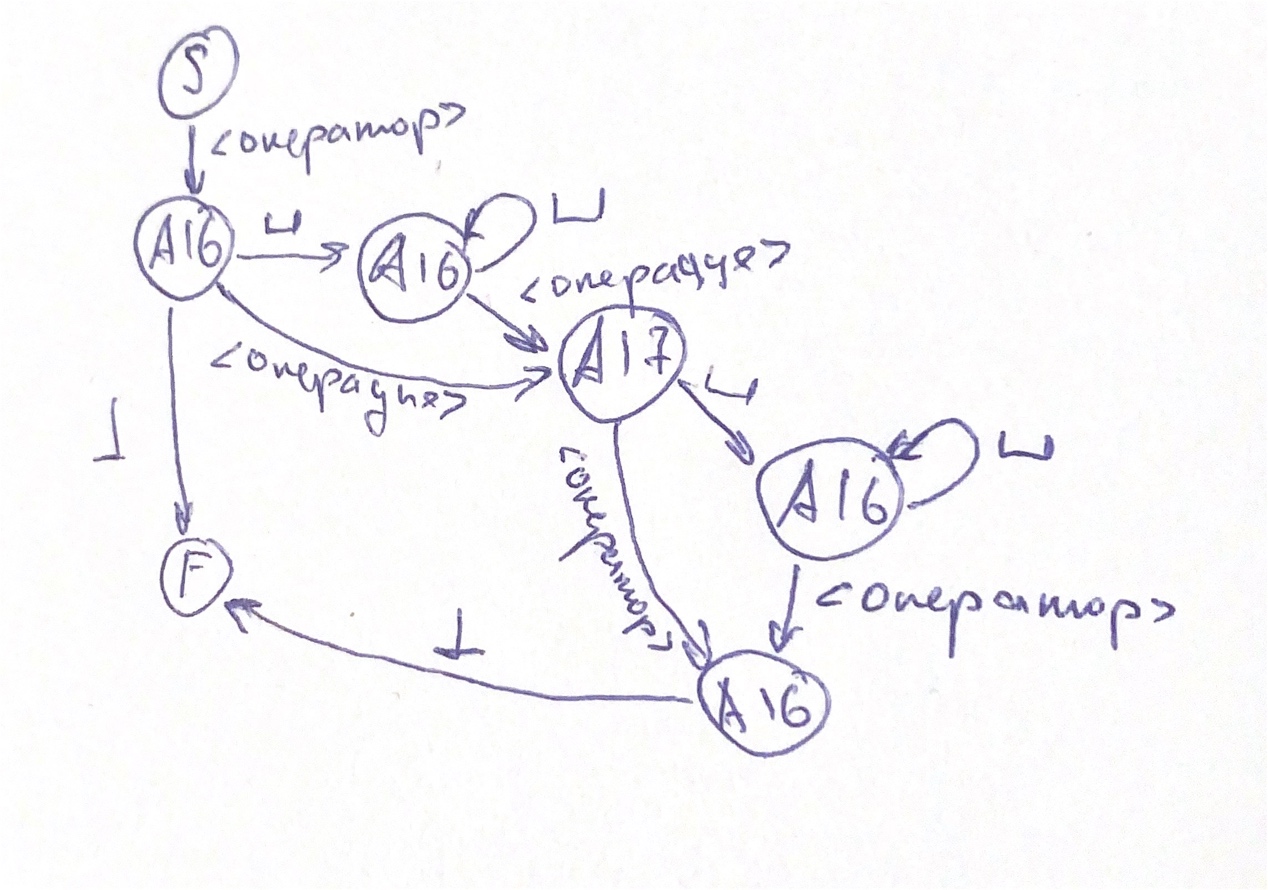


Рисунок 5. Граф состояния анализатора <правая часть>

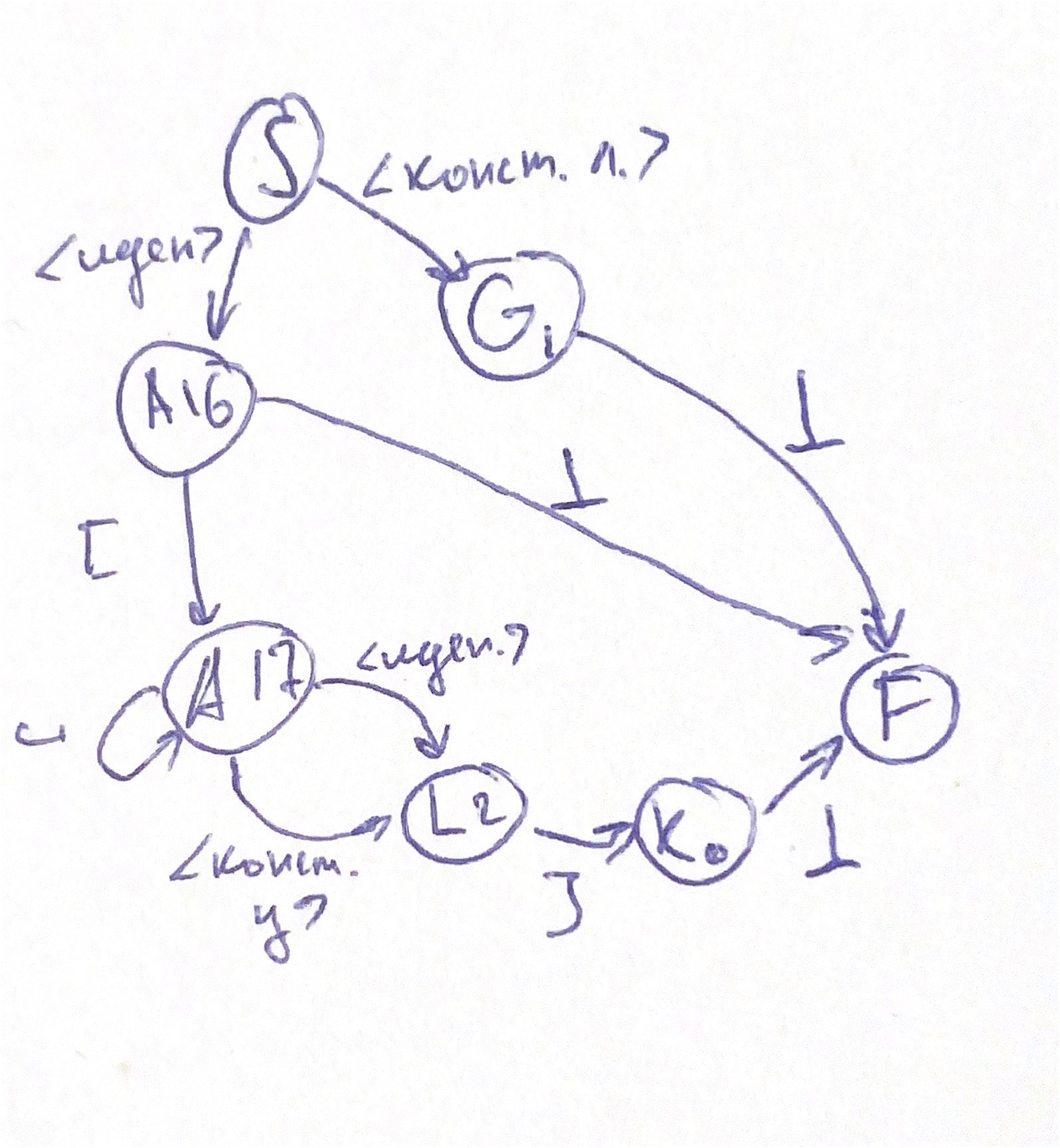


Рисунок 6. Граф состояния анализатора <оператор>

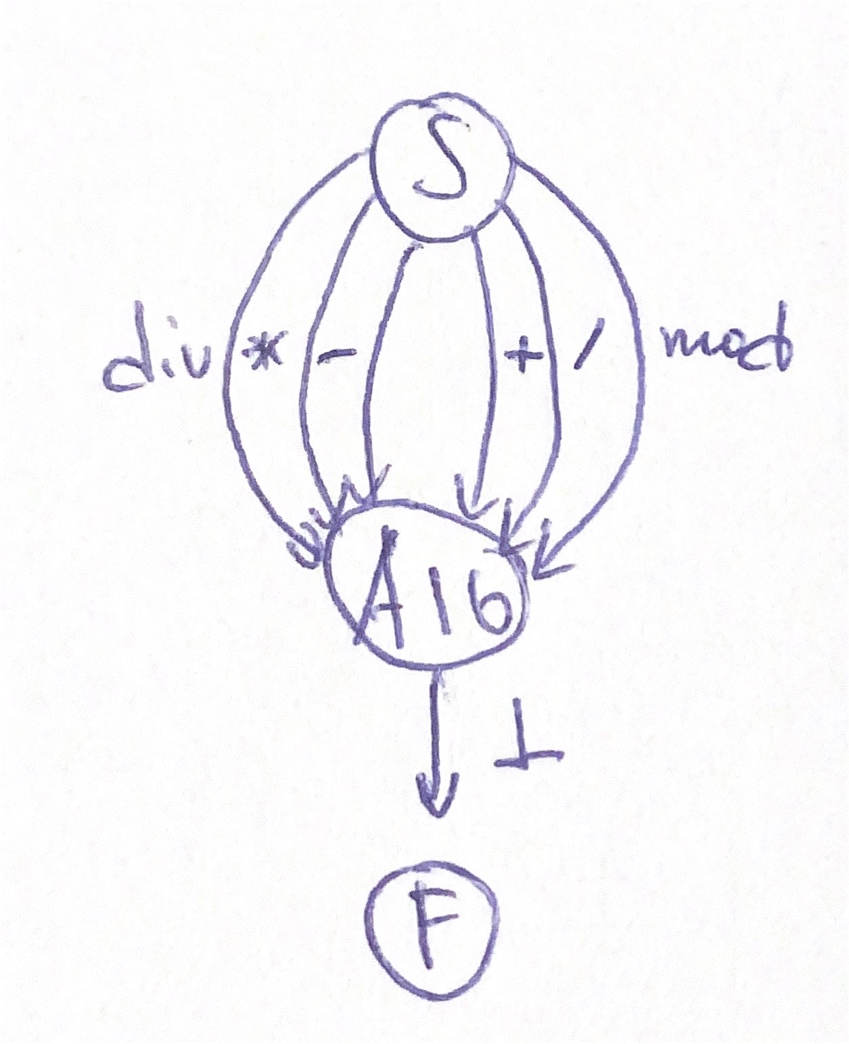


Рисунок 7. Граф состояния анализатора <операция>

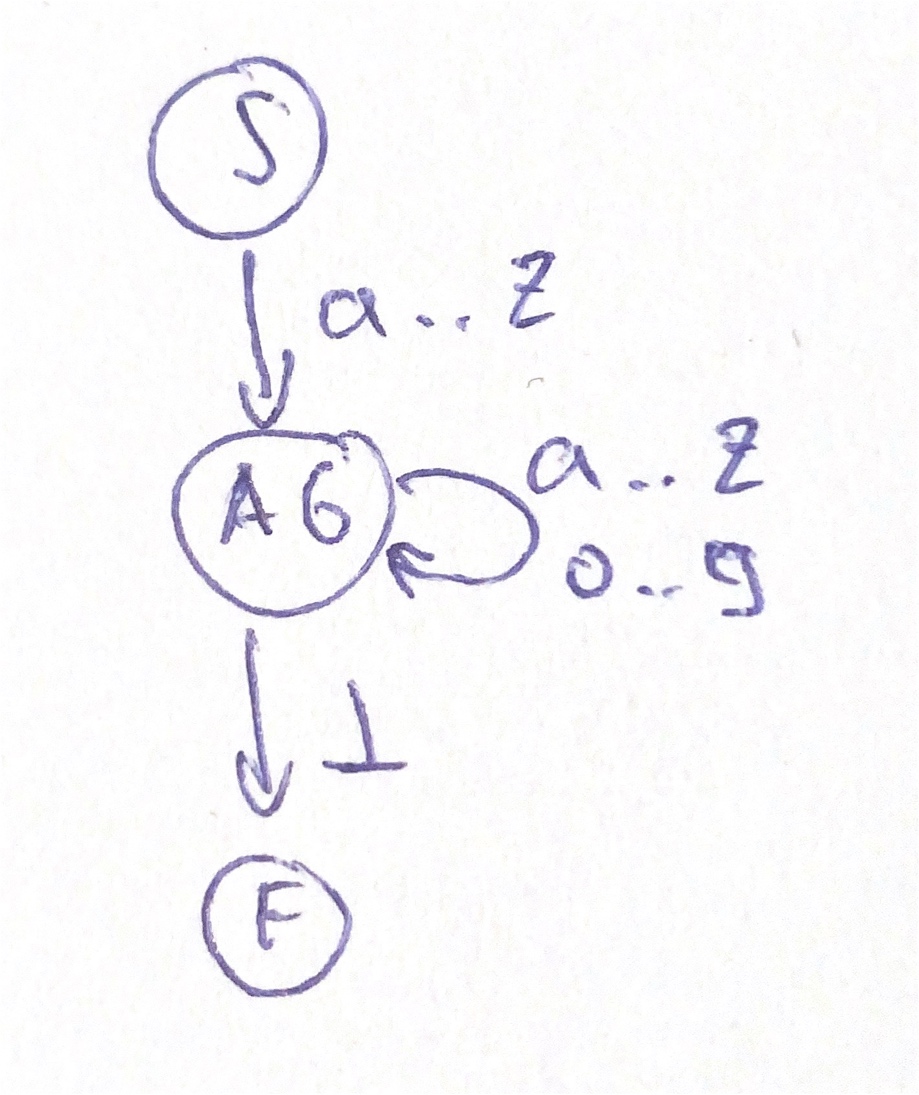


Рисунок 8. Граф состояния анализатора <идентификатор>

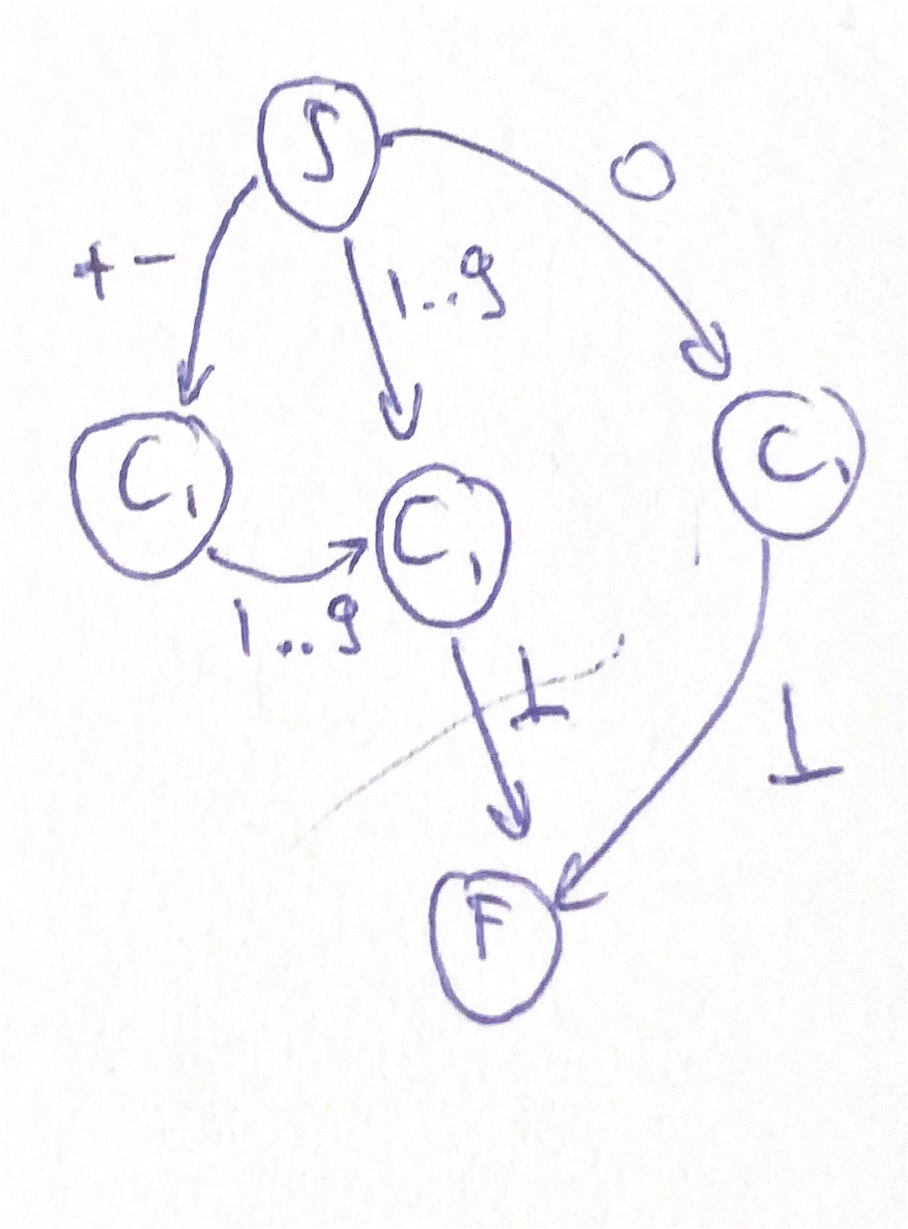


Рисунок 9. Граф состояния анализатора <константа целая>

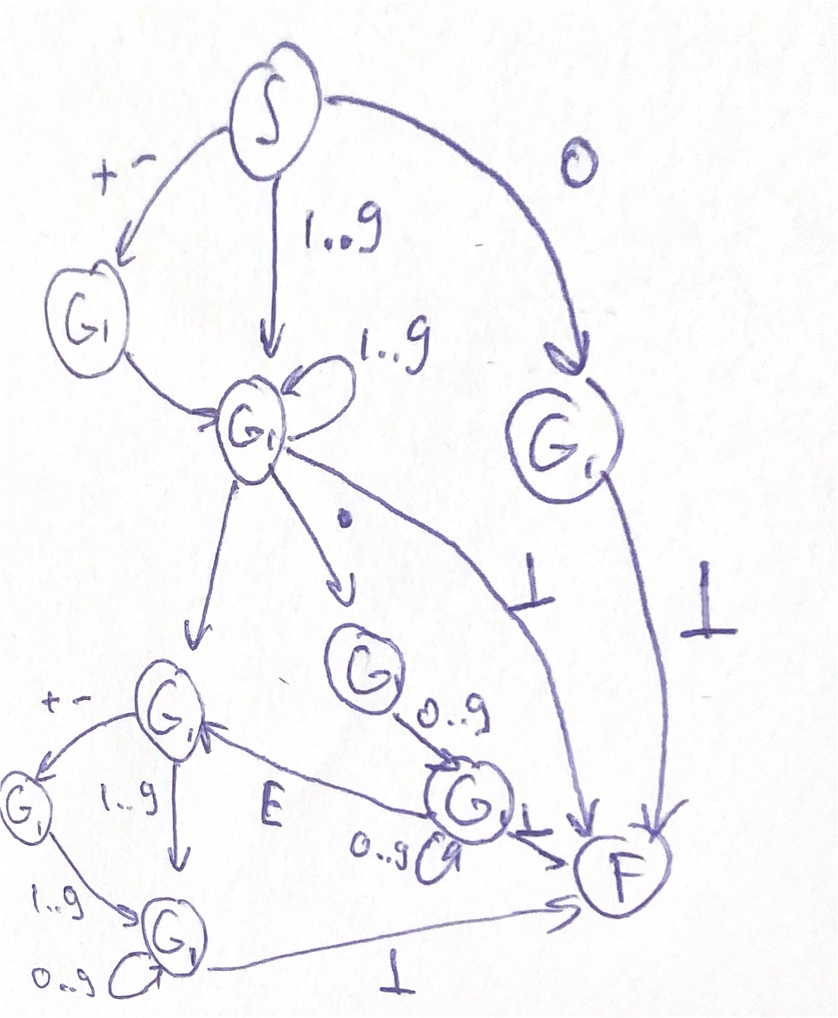


Рисунок 10. Граф состояния анализатора <константа целая>

ИНТЕРФЕЙС АНАЛИЗАТОРА

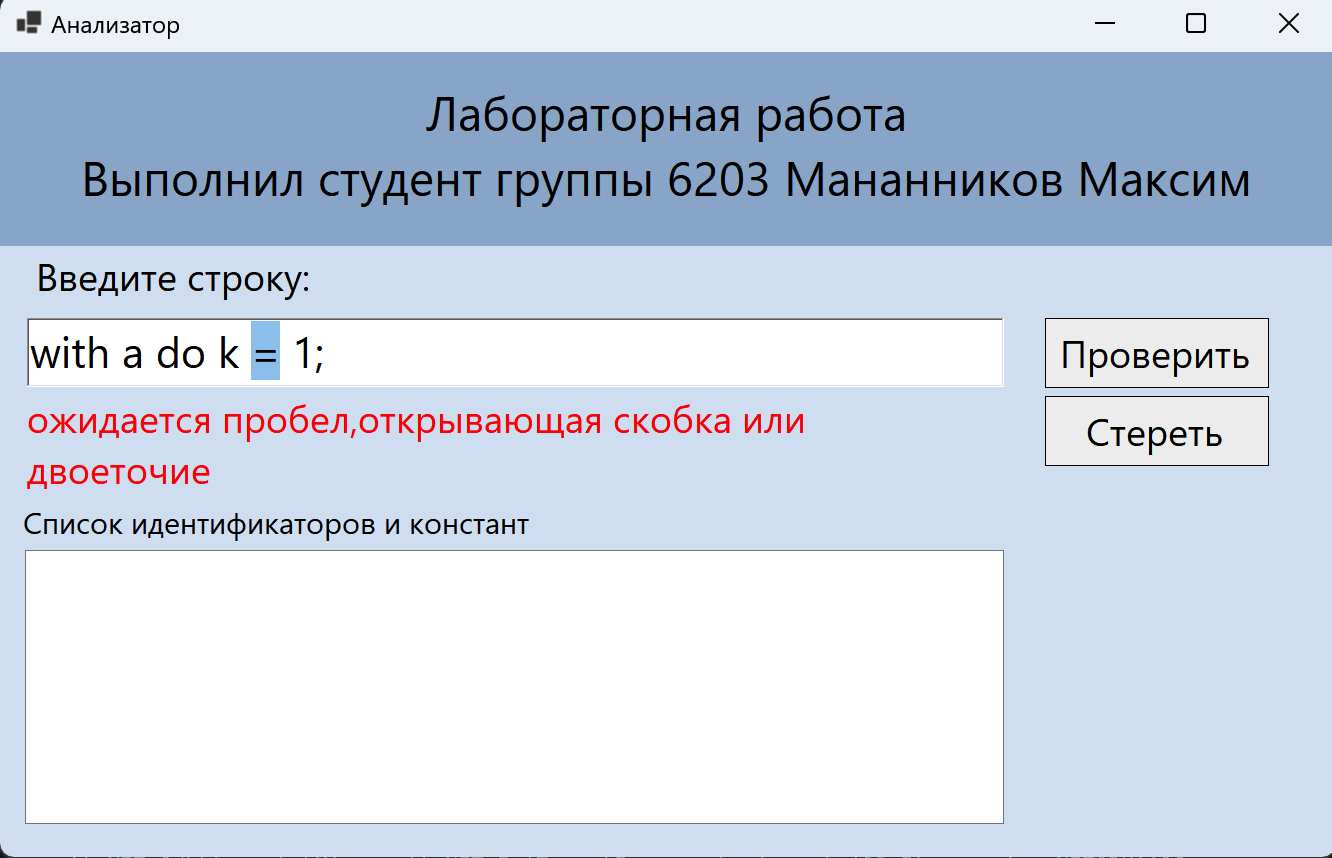


Рисунок 1. Пример синтаксической ошибки 1

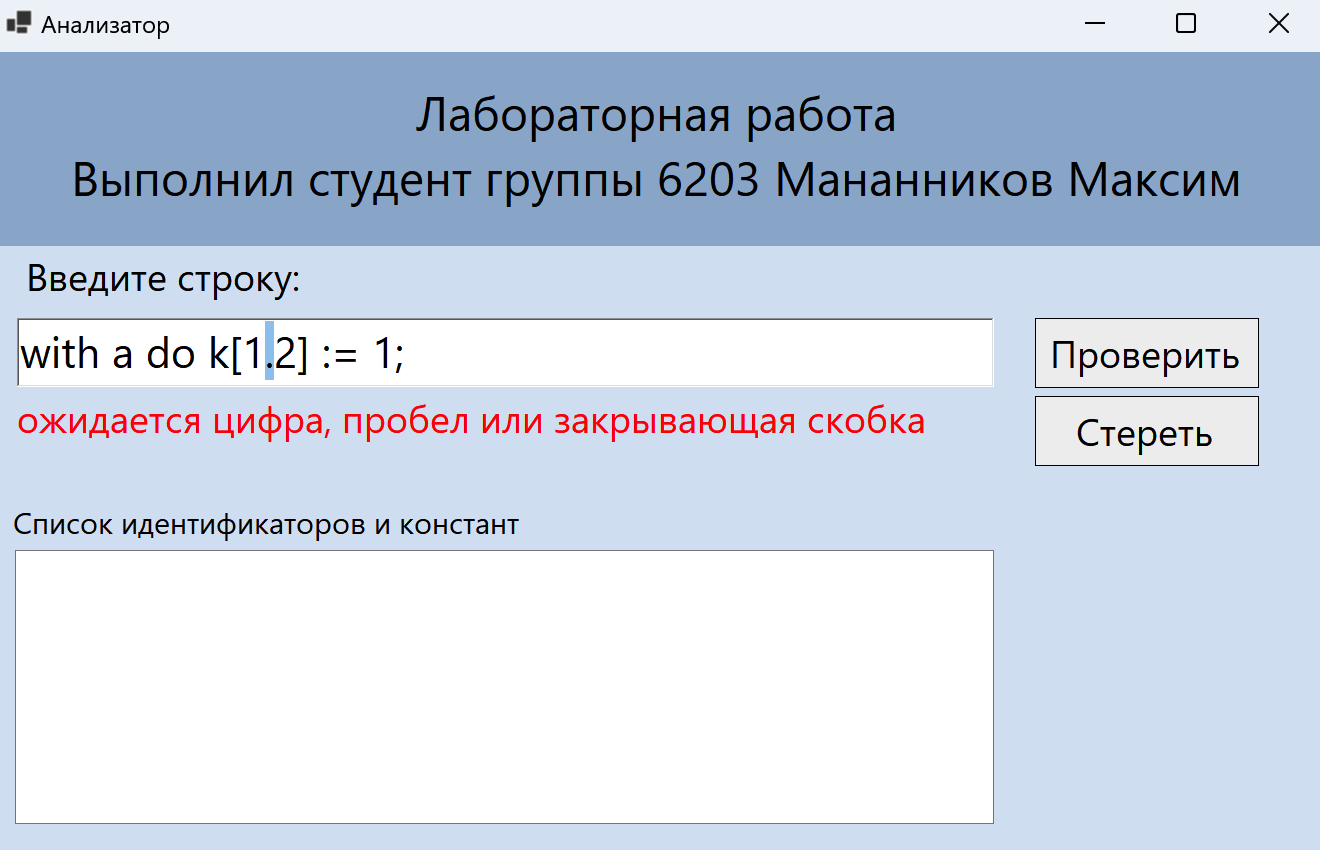


Рисунок 2. Пример синтаксической ошибки 2

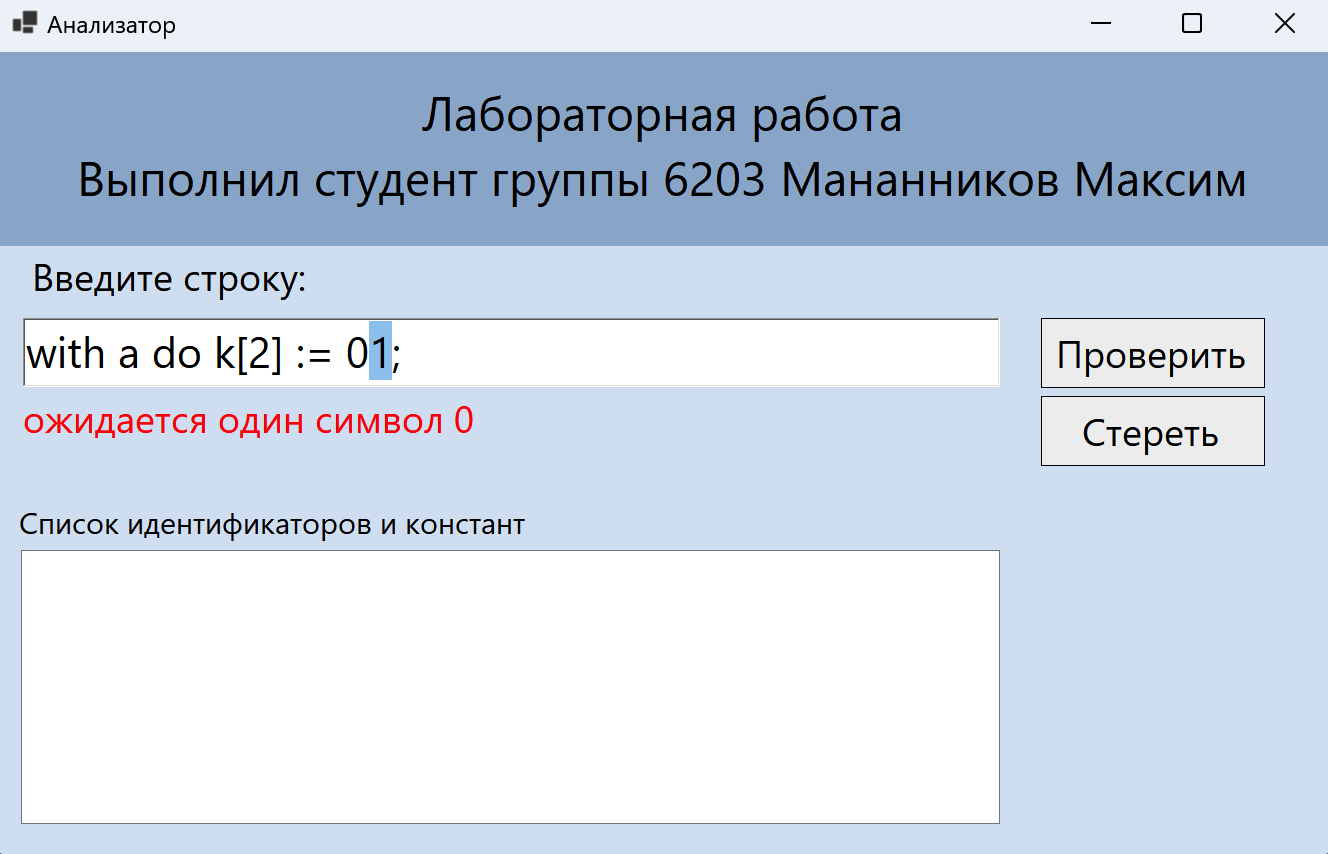


Рисунок 3. Пример синтаксической ошибки 3

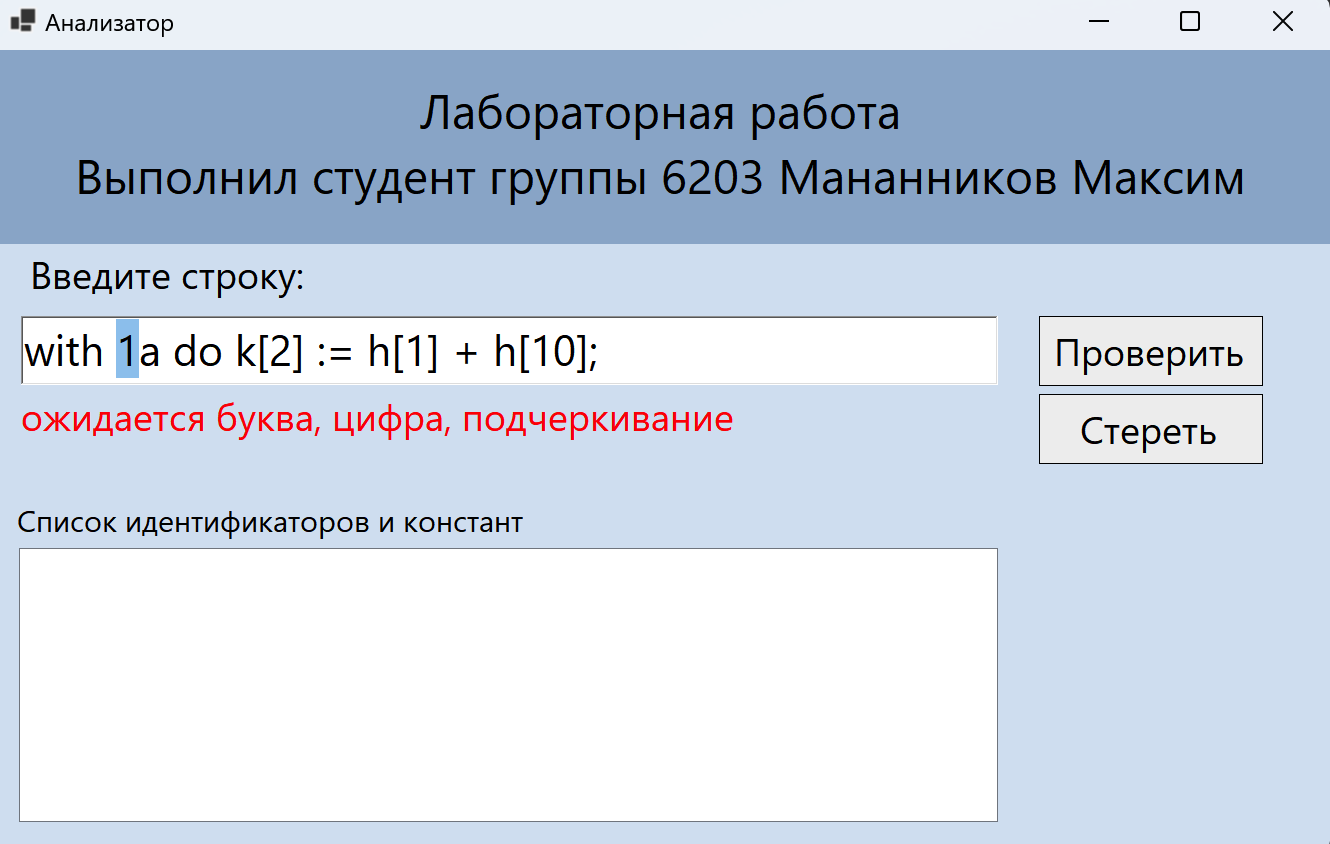


Рисунок 4. Пример синтаксической ошибки 4

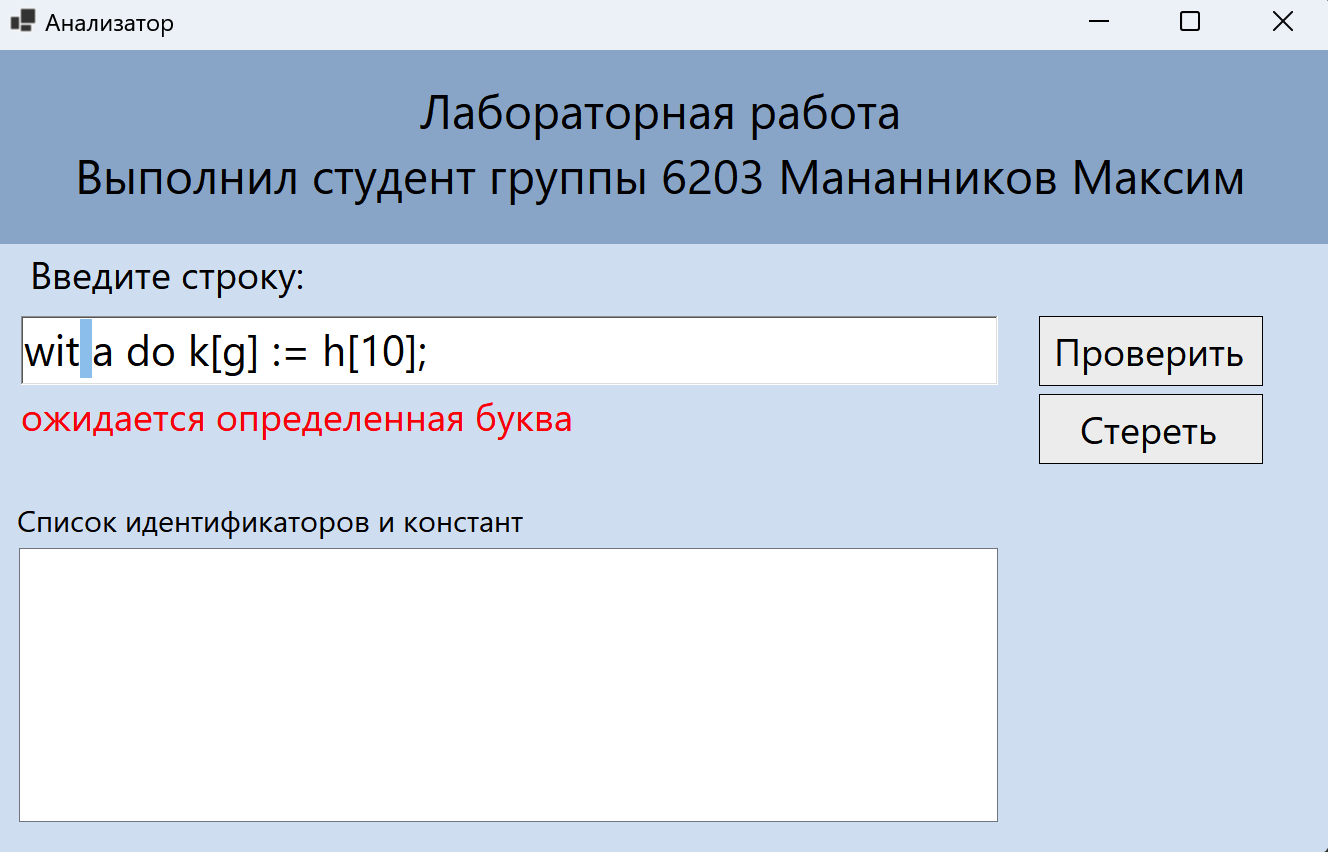


Рисунок 5. Пример синтаксической ошибки 5

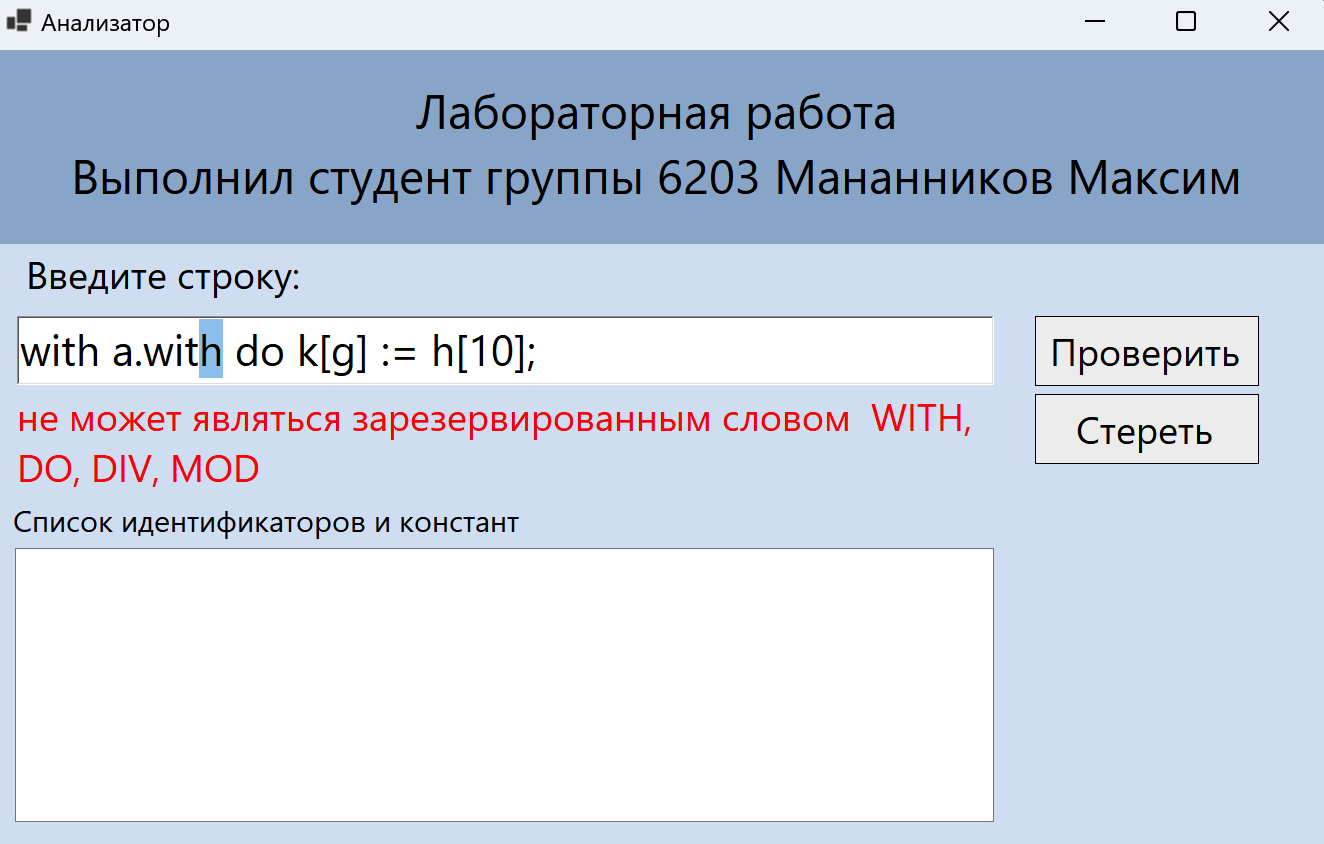


Рисунок 6. Пример семантической ошибки 1

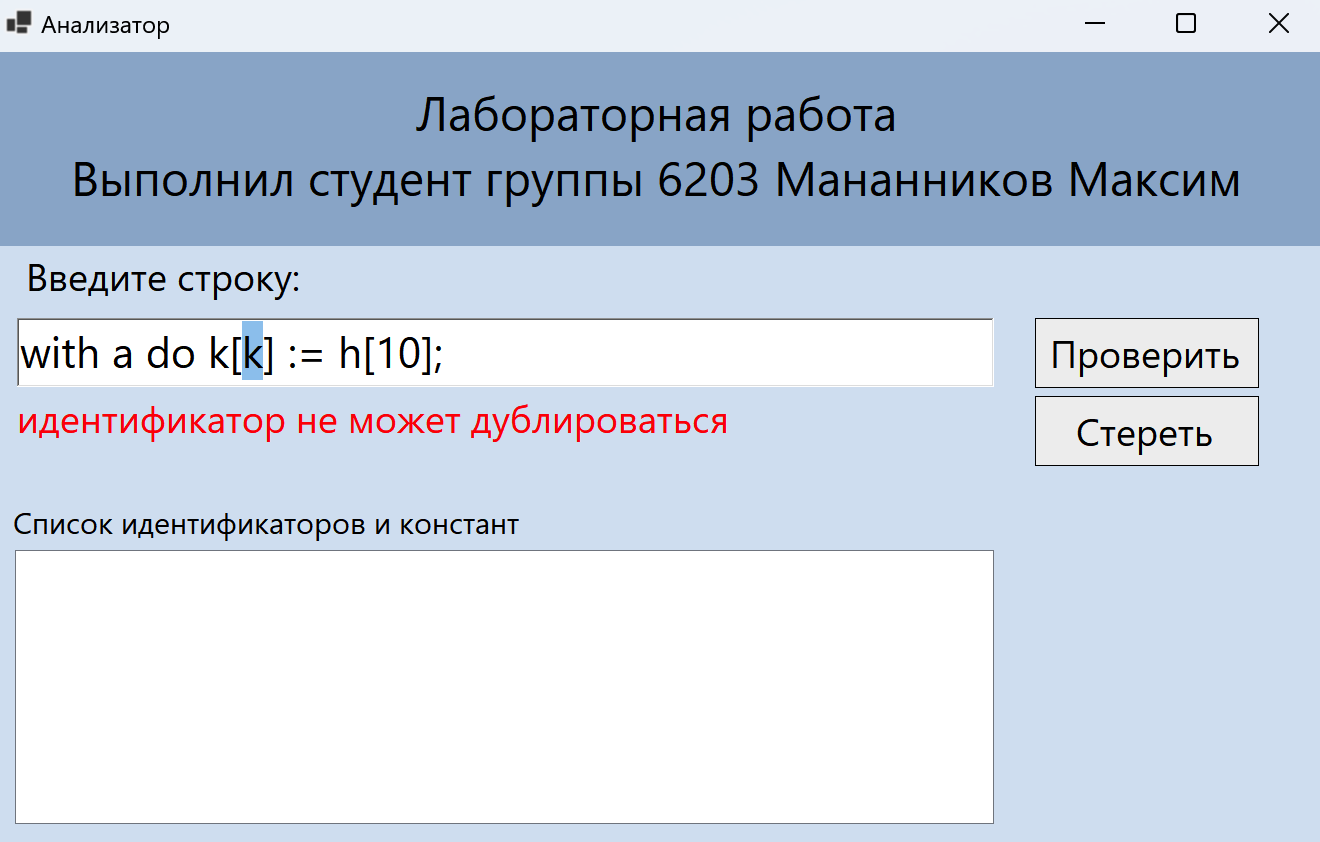


Рисунок 7. Пример семантической ошибки 2

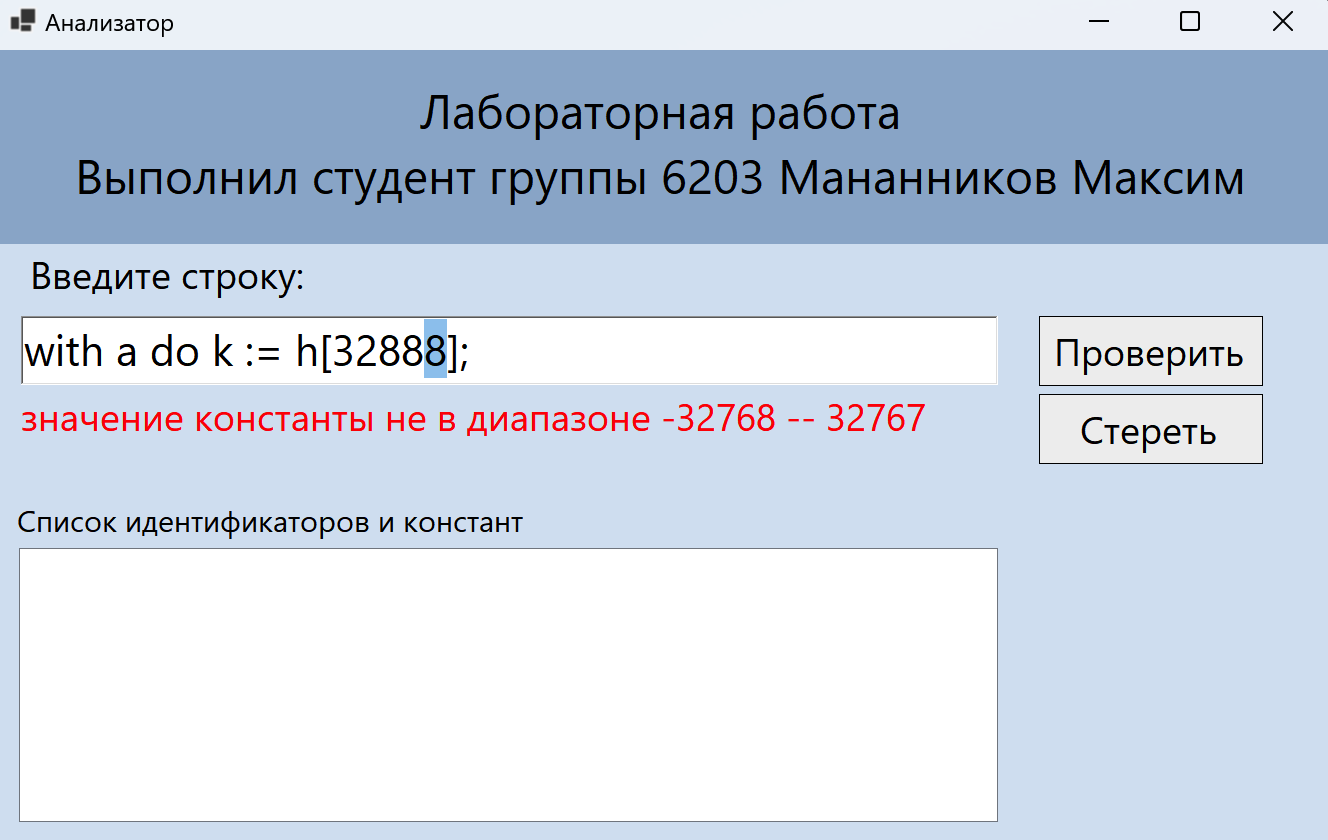


Рисунок 8. Пример семантической ошибки 3

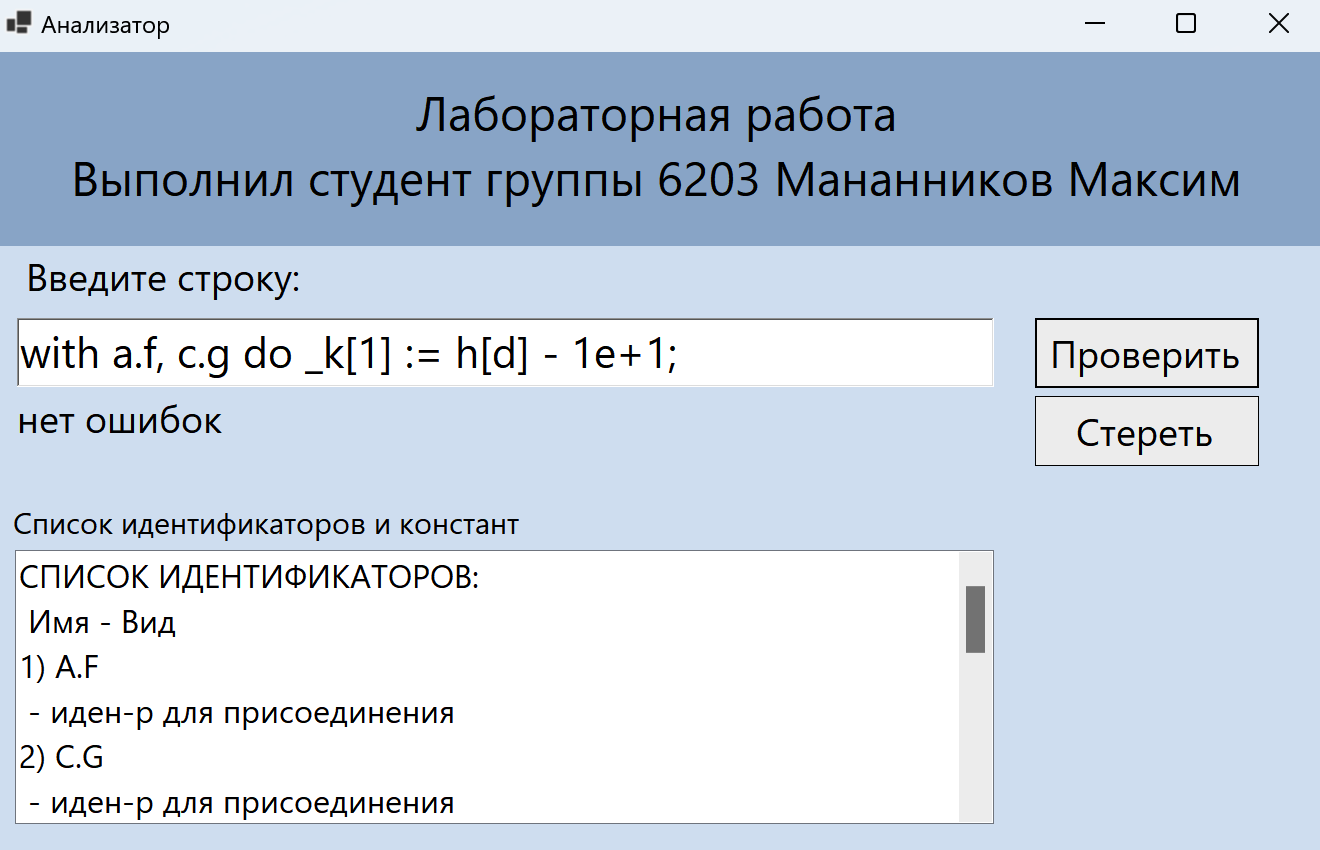


Рисунок 9. Пример вывода семантики 1

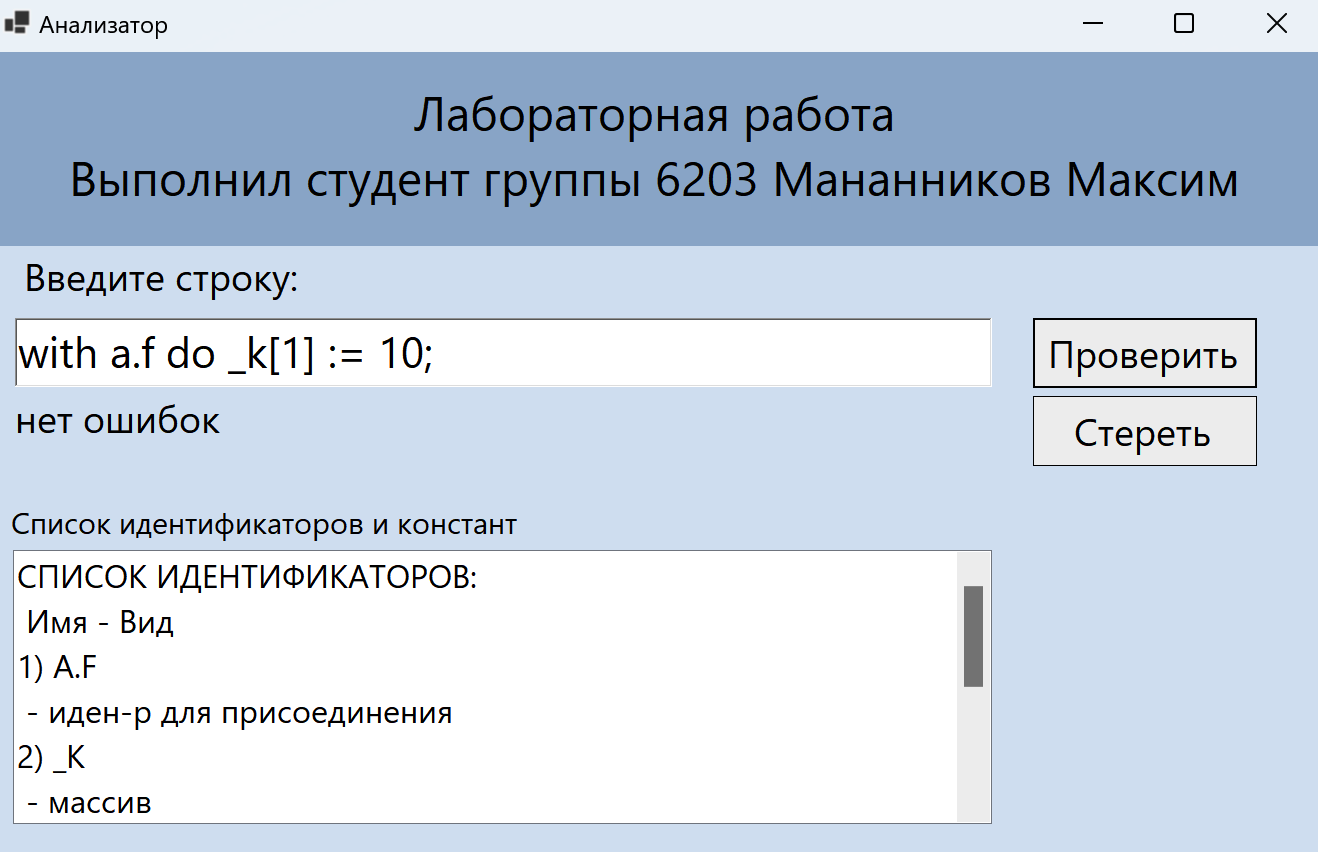


Рисунок 10. Пример вывода семантики 2

ЛИСТИНГ ОСНОВНОЙ ПРОГРАММЫ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_taifya

{

static class ClassCheck

{

// with o.r, c.f do a[a] := a[b] + a[10];

// with a.d, c.m do \_a\_n[-3] := 3e+1 - \_d1[i];

// with a do k[1] := 10.1 + 10.1;

// Перечисление состояний автомата

private enum EnumState { Start, Error, Final, a1, a2, a3, a4, a5, a6, b1, b2, a8, a9, a10, a11, a12, a13, c1, c2, c3, c4, d1, d2, a14, a16, a17, a18, a19, a20, e1, e2, g1, g2, g3, g4, g5, g6, g7, k1, k2, k0, l1, l2, x1, x2, x3, y2, y3, x4 };

// Текущая позиция курсора в анализируемой строке

private static int pos;

// Анализируемая строка

private static string str;

// Ошибка анализа

private static Err \_Err;

// Позиция курсора ошибки в анализируемой строке

private static int \_ErrPos;

// Список id

private static LinkedList<string> list\_of\_id;

// Список констант

private static LinkedList<string> list\_of\_const;

// Функция, реализующая проверку

public static ClassResult Check(string InputString)

{

pos = 0;

str = InputString;

list\_of\_id = new LinkedList<string>();

list\_of\_const = new LinkedList<string>();

SetError(Err.NoError, -1);

Realise();

return new ClassResult(\_ErrPos, \_Err, list\_of\_id, list\_of\_const);

}

// Установка типа и позиции ошибки

private static void SetError(Err ErrorType, int ErrorPosition)

{

\_Err = ErrorType;

\_ErrPos = ErrorPosition;

}

private static bool Realise()

{

int i = 1; // счетчик букв

int m = 1; // счетчик цифр

int id\_count = 1;

int const\_count = 1;

string mass = ""; // временная переменная для проверки названия массива

string mass\_1 = ""; // для проверки после знака =

string consta = ""; // временная переменная для проверки константы целой

string consta\_1 = ""; // для проверки константы с фикс. точкой

string index = ""; // для проверки индекса

string index\_1 = ""; // для проверки индекса после =

// Состояние автомата

EnumState State = EnumState.Start;

string idName = "";

string IdNumber = "";

// Основной цикл ДКА

while ((State != EnumState.Error) && (State != EnumState.Final))

{

// Если вышли за границы строки

if (pos >= str.Length)

{

SetError(Err.OutOfRange, pos - 1);

State = EnumState.Error;

}

else

{

// Проверяем состояние ДКА

switch (State)

{

// ожидается пробел или W

case EnumState.Start:

{

if (str[pos] == 'W')

State = EnumState.a1;

else if (str[pos] == ' ')

State = EnumState.Start;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается I

case EnumState.a1:

{

if (str[pos] == 'I')

State = EnumState.a2;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается T

case EnumState.a2:

{

if (str[pos] == 'T')

State = EnumState.a3;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается H

case EnumState.a3:

{

if (str[pos] == 'H')

State = EnumState.a4;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел

case EnumState.a4:

{

if (str[pos] == ' ')

State = EnumState.a5;

else

{

SetError(Err.SpaceExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел, нижнее подчеркивание или буква

case EnumState.a5:

{

if (str[pos] == ' ')

State = EnumState.a5;

else if (char.IsLetter(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i = 1; // будем накапливать буквы

State = EnumState.a6;

}

else

{

SetError(Err.LetterDigitUnderscoreExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел, нижнее подчеркивание, цифра, точка или запятая

case EnumState.a6:

{

if (str[pos] == ',')

State = EnumState.a5;

else if (str[pos] == '.')

{

idName += str[pos];

i = 1;

State = EnumState.a6;

}

else if (str[pos] == ' ')

State = EnumState.a8;

else if (char.IsLetterOrDigit(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i++;

State = EnumState.a6;

}

else

{

SetError(Err.LetterDigitUnderscoreSpaceCommaDotExpected, pos);

State = EnumState.Error;

}

if (i > 8)

{

SetError(Err.OverflowIdSymbols, pos);

State = EnumState.Error;

}

if ((State == EnumState.a5) || (State == EnumState.b1) || (State == EnumState.a8))

{

idName.ToUpper();

if ((idName.Contains(".WITH")) || (idName.Contains(".DO")) || (idName.Contains(".DIV")) || (idName.Contains(".MOD")))

{

if (idName.Contains(".") && idName.Contains("WITH"))

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DO"))

{

SetError(Err.IncorrectId, pos - 2);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DIV"))

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("MOD"))

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

else

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

}

else if ((idName.Contains("WITH")) || (idName.Contains("DO")) || (idName.Contains("DIV")) || (idName.Contains("MOD")))

{

if (idName.Contains(".") && idName.Contains("WITH"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DO"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DIV"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("MOD"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

}

else

{

idName = id\_count.ToString() + ") " + idName;

list\_of\_id.AddLast(idName);

list\_of\_id.AddLast(" - иден-р для присоединения");

id\_count++;

idName = "";

}

}

break;

}

// ожидается буква или нижнее подчеркивание

case EnumState.b1:

{

if (char.IsLetter(str[pos]) || (str[pos] == ' '))

{

idName += str[pos];

i = 1; // будем накапливать буквы

State = EnumState.b2;

}

else

{

SetError(Err.LetterUnderscoreExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается буква, цифра, нижнее подчеркивание, запятая или пробел

case EnumState.b2:

{

if (str[pos] == ',')

State = EnumState.a5;

else if (str[pos] == ' ')

State = EnumState.a8;

else if (char.IsLetterOrDigit(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i++;

State = EnumState.b2;

}

else

{

SetError(Err.LetterDigitUnderscoreSpaceCommaDotExpected, pos);

State = EnumState.Error;

}

if (i > 8)

{

SetError(Err.OverflowIdSymbols, pos);

State = EnumState.Error;

}

if ((State == EnumState.a5) || (State == EnumState.a8))

{

idName.ToUpper();

if ((idName.Contains(".WITH")) || (idName.Contains(".DO")) || (idName.Contains(".DIV")) || (idName.Contains(".MOD")))

{

if (idName.Contains(".") && idName.Contains("WITH"))

{

SetError(Err.IncorrectId, pos);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DO"))

{

SetError(Err.IncorrectId, pos);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DIV"))

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("MOD"))

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

else

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

}

else if ((idName.Contains("WITH")) || (idName.Contains("DO")) || (idName.Contains("DIV")) || (idName.Contains("MOD")))

{

if (idName.Contains(".") && idName.Contains("WITH"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DO"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("DIV"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else if (idName.Contains(".") && idName.Contains("MOD"))

{

SetError(Err.IncorrectId, pos - 3);

State = EnumState.Error;

}

else

{

SetError(Err.IncorrectId, pos - 1);

State = EnumState.Error;

}

}

else

{

idName = id\_count.ToString() + ") " + idName;

list\_of\_id.AddLast(idName);

list\_of\_id.AddLast(" - иден-р для присоединения");

id\_count++;

idName = "";

}

}

break;

}

//ожидается пробел, D или запятая

case EnumState.a8:

{

//LetterSpaceCommaExpected

if (str[pos] == ',')

State = EnumState.a5;

else if (str[pos] == 'D')

State = EnumState.a9;

else if (str[pos] == ' ')

State = EnumState.a8;

else

{

SetError(Err.LetterSpaceCommaExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается O

case EnumState.a9:

{

if (str[pos] == 'O')

State = EnumState.a10;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел

case EnumState.a10:

{

if (str[pos] == ' ')

State = EnumState.a11;

else

{

SetError(Err.SpaceExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел, буква или нижнее подчеркивание

case EnumState.a11:

{

if (str[pos] == ' ')

State = EnumState.a11;

else if (char.IsLetter(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i = 1; // будем накапливать буквы

State = EnumState.a12;

}

else

{

SetError(Err.LetterUnderscoreSpaceExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается буква, нижнее подчеркивание, пробел, двоеточие или открывающаяся скобка

case EnumState.a12:

{

if (str[pos] == ' ')

State = EnumState.a13;

else if (str[pos] == '[')

State = EnumState.c1;

else if (str[pos] == ':')

State = EnumState.a14;

else if (char.IsLetterOrDigit(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i++;

State = EnumState.a12;

}

else

{

SetError(Err.LetterDigitSpaceUnderscoreСolonOpenbracketExpected, pos);

State = EnumState.Error;

}

if (i >= 8)

{

SetError(Err.OverflowIdSymbols, pos);

State = EnumState.Error;

}

if ((State == EnumState.a13) || (State == EnumState.a14) || (State == EnumState.c1))

{

idName.ToUpper();

if ((idName == "WITH") || (idName == "DO") || (idName == "DIV") || (idName == "MOD"))

{

SetError(Err.IncorrectId, pos);

State = EnumState.Error;

}

else

{

mass = idName;

idName = id\_count.ToString() + ") " + idName;

list\_of\_id.AddLast(idName);

list\_of\_id.AddLast(" - массив");

id\_count++;

idName = "";

}

}

break;

}

// ожидается пробел, открывающая скобка или двоеточие

case EnumState.a13:

{

if (str[pos] == ':')

State = EnumState.a14;

else if (str[pos] == '[')

State = EnumState.c1;

else if (str[pos] == ' ')

State = EnumState.a13;

else

{

SetError(Err.SpaceСolonOpenbracketExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел, буквы, нижнее подчеркивание, цифры или минус

case EnumState.c1:

{

if (str[pos] == ' ')

State = EnumState.c1;

else if (str[pos] == '-')

{

IdNumber += str[pos];

m = 1;

State = EnumState.d1;

}

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m = 1;

State = EnumState.d2;

}

else if (char.IsLetter(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i = 1;

State = EnumState.c2;

}

else

{

SetError(Err.LetterDigitUnderscoreMinusSpaceExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается цифра

case EnumState.d1:

{

if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.d2;

}

else

{

SetError(Err.DigitExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается буква, цифра, нижнее подчеркивание, пробел или закрывающая скобка

case EnumState.c2:

{

if (str[pos] == ' ')

State = EnumState.c3;

else if (str[pos] == ']')

State = EnumState.c4;

else if (char.IsLetterOrDigit(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i++;

State = EnumState.c2;

}

else

{

SetError(Err.LetterDigitUnderscoreSpaceSemicolonCloseBracketExpected, pos);

State = EnumState.Error;

}

if (i > 8)

{

SetError(Err.OverflowIdSymbols, pos);

State = EnumState.Error;

}

if ((State == EnumState.c3) || (State == EnumState.c4))

{

idName.ToUpper();

if ((idName == "WITH") || (idName == "DO") || (idName == "DIV") || (idName == "MOD"))

{

SetError(Err.IncorrectId, pos);

State = EnumState.Error;

}

else

{

if (idName != mass)

{

idName = id\_count.ToString() + ") " + idName;

list\_of\_id.AddLast(idName);

list\_of\_id.AddLast(" - индекс");

id\_count++;

idName = "";

}

else

{

SetError(Err.DublicateError, pos - 1);

State = EnumState.Error;

}

}

}

break;

}

// ожидается цифра, пробел или закрывающая скобка

case EnumState.d2:

{

if (str[pos] == ' ')

State = EnumState.c3;

else if (str[pos] == ']')

State = EnumState.c4;

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.d2;

}

else

{

SetError(Err.DigitSpaceClosebracket, pos);

State = EnumState.Error;

}

if ((State == EnumState.c3) || (State == EnumState.c4))

{

if (Int16.TryParse(IdNumber.Substring(0), out Int16 val))

{

if(!list\_of\_const.Contains(" - индекс - целая"))

{

index = IdNumber;

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - индекс - целая");

const\_count++;

IdNumber = "";

}

}

else

{

SetError(Err.ValueUotOfRange, pos - 1);

State = EnumState.Error;

}

}

break;

}

// ожидается пробел или закрывающая скобка

case EnumState.c3:

{

if (str[pos] == ' ')

State = EnumState.c3;

else if (str[pos] == ']')

State = EnumState.c4;

else

{

SetError(Err.SpaceClosebracket, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел или двоеточие

case EnumState.c4:

{

if (str[pos] == ' ')

State = EnumState.c4;

else if (str[pos] == ':')

State = EnumState.a14;

else

{

SetError(Err.SpaceColonExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается =

case EnumState.a14:

{

if (str[pos] == '=')

State = EnumState.a16;

else

{

SetError(Err.Equal, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел, минус, цифра, буква, нижнее подчеркивание или одинарная кавычка

case EnumState.a16:

{

if (str[pos] == ' ')

State = EnumState.a16;

else if (str[pos] == '-')

{

IdNumber += str[pos];

m = 1;

State = EnumState.g2;

}

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m = 1;

State = EnumState.g1;

}

else if (str[pos] == '\'')

State = EnumState.e1;

else if (char.IsLetter(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i = 1;

State = EnumState.a17;

}

else

{

SetError(Err.LetterDigitUnderscoreSpaceMinusMark, pos);

State = EnumState.Error;

}

break;

}

// ожидается символ

case EnumState.e1:

{

if (((int)str[pos] > 32) && (str[pos] != '\''))

State = EnumState.e2;

else if (str[pos] == '\'')

State = EnumState.a20;

else

{

SetError(Err.NotManagerMark, pos);

State = EnumState.Error;

}

break;

}

// ожидается неуправляющий символ или кавычка

case EnumState.e2:

{

if (((int)str[pos] > 32) && (str[pos] != '\''))

State = EnumState.e2;

else if (str[pos] == '\'')

State = EnumState.a20;

else

{

SetError(Err.NotManagerMark, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел или точка с запятой

case EnumState.a20:

{

if (str[pos] == ' ')

State = EnumState.a20;

else if (str[pos] == ';')

State = EnumState.Final;

else

{

SetError(Err.SpaceSemicolon, pos);

State = EnumState.Error;

}

break;

}

// ожидается цифра

case EnumState.g2:

{

if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.g1;

}

else

{

SetError(Err.DigitExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается мат операция, цифры, точка, e, ;, пробел

case EnumState.g1:

{

if (str[pos] == '.')

{

IdNumber += str[pos];

m++;

State = EnumState.g3;

}

else if ((str[pos] == 'e') || (str[pos] == 'E'))

{

IdNumber += str[pos];

m++;

State = EnumState.g5;

}

else if (str[pos] == ';')

State = EnumState.Final;

else if (str[pos] == ' ')

State = EnumState.x1;

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.g1;

}

else if ((str[pos] == '+') || (str[pos] == '-') || (str[pos] == '\*') || (str[pos] == '/'))

State = EnumState.a16;

else

{

SetError(Err.DigitDotMathOperESemicolonSpaceExpected, pos);

State = EnumState.Error;

}

if ((State == EnumState.Final) || (State == EnumState.x1) || (State == EnumState.a16))

{

if (IdNumber[0] == '0' && IdNumber.Length > 1)

{

SetError(Err.InsignificantZero, pos - 1);

State = EnumState.Error;

}

else if (Int16.TryParse(IdNumber.Substring(0), out Int16 val))

{

if (!list\_of\_const.Contains(" - константа выражения - целая"))

{

consta = IdNumber;

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - константа выражения - целая");

const\_count++;

IdNumber = "";

}

else

{

if (IdNumber != consta)

{

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - константа выражения - целая");

const\_count++;

IdNumber = "";

}

IdNumber = "";

}

}

else

{

SetError(Err.ValueUotOfRange, pos - 1);

State = EnumState.Error;

}

}

break;

}

// ожидается цифра

case EnumState.g3:

{

if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.g4;

}

else

{

SetError(Err.DigitExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается цифра, мат опоерация, e, пробел или ;

case EnumState.g4:

{

if ((str[pos] == '+') || (str[pos] == '-') || (str[pos] == '\*') || (str[pos] == '/'))

State = EnumState.a16;

else if ((str[pos] == 'e') || str[pos] == 'E')

{

IdNumber += str[pos];

m++;

State = EnumState.g5;

}

else if (str[pos] == ' ')

State = EnumState.x1;

else if (str[pos] == ';')

State = EnumState.Final;

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.g4;

}

else

{

SetError(Err.DigitSpaceSemicolonMathOperEExpected, pos - 1);

State = EnumState.Error;

}

if ((State == EnumState.Final) || (State == EnumState.x1) || (State == EnumState.a16))

{

if(!list\_of\_const.Contains(" - константа выражения - с фикс. точкой"))

{

consta\_1 = IdNumber;

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - константа выражения - с фикс. точкой");

const\_count++;

IdNumber = "";

}

else

{

if(IdNumber != consta\_1)

{

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - константа выражения - с фикс. точкой");

const\_count++;

IdNumber = "";

}

IdNumber = "";

}

}

break;

}

// ожидается цифра или минус

case EnumState.g5:

{

if (str[pos] == '-' || str[pos] == '+')

{

IdNumber += str[pos];

m++;

State = EnumState.g7;

}

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.g6;

}

else

{

SetError(Err.DigitExpected, pos - 1);

State = EnumState.Error;

}

break;

}

// ожидается цифра

case EnumState.g7:

{

if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.g6;

}

else

{

SetError(Err.DigitExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается цифра, мат операция, пробел или точка с запятой

case EnumState.g6:

{

if ((str[pos] == '+') || (str[pos] == '-') || (str[pos] == '\*') || (str[pos] == '/'))

State = EnumState.a16;

else if (str[pos] == ' ')

State = EnumState.x1;

else if (str[pos] == ';')

State = EnumState.Final;

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.g6;

}

else

{

SetError(Err.DigitMathOperSemicolonSpace, pos - 1);

State = EnumState.Error;

}

if ((State == EnumState.x1) || (State == EnumState.Final) || (State == EnumState.a16))

{

if(!list\_of\_const.Contains(" - константа выражения - с плав. точкой"))

{

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - константа выражения - с плав. точкой");

const\_count++;

IdNumber = "";

}

}

break;

}

// ожидается буква, цифра, нижнее подчеркивание, мат операция, пробел, открывающая скобка, точка с запятой

case EnumState.a17:

{

if ((str[pos] == '+') || (str[pos] == '-') || (str[pos] == '\*') || (str[pos] == '/'))

State = EnumState.a16;

else if (str[pos] == ' ')

State = EnumState.a18;

else if (str[pos] == '[')

State = EnumState.k1;

else if (str[pos] == ';')

State = EnumState.Final;

else if (char.IsLetterOrDigit(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i++;

State = EnumState.a17;

}

else

{

SetError(Err.LetterDigitUnderscopeMathOperSpaceOpenbracketSemicolon, pos - 1);

State = EnumState.Error;

}

if (i > 8)

{

SetError(Err.OverflowIdSymbols, pos - 1);

State = EnumState.Error;

}

if ((State == EnumState.a18) || (State == EnumState.k1) || (State == EnumState.a16) || (State == EnumState.Final))

{

idName.ToUpper();

if ((idName == "WITH") || (idName == "DO") || (idName == "DIV") || (idName == "MOD"))

{

SetError(Err.IncorrectId, pos);

State = EnumState.Error;

}

else

{

if (idName != mass)

{

if(idName != mass\_1)

{

mass\_1 = idName;

idName = id\_count.ToString() + ") " + idName;

list\_of\_id.AddLast(idName);

list\_of\_id.AddLast(" - массив");

id\_count++;

idName = "";

}

else

{

idName = "";

}

}

idName = "";

}

}

break;

}

// ожидается пробел, мат операция, D, M, открывающая скобка или точка с запятой

case EnumState.a18:

{

if (str[pos] == ' ')

State = EnumState.a18;

else if ((str[pos] == '+') || (str[pos] == '-') || (str[pos] == '\*') || (str[pos] == '/'))

State = EnumState.a16;

else if (str[pos] == '[')

State = EnumState.k1;

else if (str[pos] == ';')

State = EnumState.Final;

else if (str[pos] == 'M')

State = EnumState.x2;

else if (str[pos] == 'D')

State = EnumState.y2;

else

{

SetError(Err.LetterMathOperSpaceOpenbracketSemicolon, pos);

State = EnumState.Error;

}

break;

}

// ожидается буква, нижнее подчеркивание, минус или цифры

case EnumState.k1:

{

if (str[pos] == ' ')

State = EnumState.k2;

else if (str[pos] == '-')

{

IdNumber += str[pos];

m = 1;

State = EnumState.l1;

}

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.l2;

}

else if (char.IsLetter(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i++;

State = EnumState.k2;

}

else

{

SetError(Err.LetterDigitUnderscopeMinus, pos - 1);

State = EnumState.Error;

}

break;

}

// ожидается цифра

case EnumState.l1:

{

if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.l2;

}

else

{

SetError(Err.DigitExpected, pos - 1);

State = EnumState.Error;

}

break;

}

// ожидается буква, цифра, ниижнее подчеркивание, пробел, закрывающая скобка

case EnumState.k2:

{

if (str[pos] == ' ')

State = EnumState.k2;

else if (str[pos] == ']')

State = EnumState.k0;

else if (char.IsLetter(str[pos]) || (str[pos] == '\_'))

{

idName += str[pos];

i++;

State = EnumState.k2;

}

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++ ;

State = EnumState.l2;

}

else

{

SetError(Err.LetterDigitUnderscopeSpaceClosebracket, pos - 1);

State = EnumState.Error;

}

if (i > 8)

{

SetError(Err.OverflowIdSymbols, pos - 1);

State = EnumState.Error;

}

if ((State == EnumState.k0) || (State == EnumState.a19))

{

idName.ToUpper();

if ((idName == "WITH") || (idName == "DO") || (idName == "DIV") || (idName == "MOD"))

{

SetError(Err.IncorrectId, pos);

State = EnumState.Error;

}

else

{

if (idName != mass\_1)

{

idName = id\_count.ToString() + ") " + idName;

list\_of\_id.AddLast(idName);

list\_of\_id.AddLast(" - индекс");

id\_count++;

idName = "";

}

else

{

SetError(Err.DublicateError, pos - 1);

State = EnumState.Error;

}

}

}

break;

}

// ожидается пробел, закрывающая скобка или цифра

case EnumState.l2:

{

if (str[pos] == ']')

State = EnumState.k0;

else if (str[pos] == ' ')

State = EnumState.a19;

else if (char.IsDigit(str[pos]))

{

IdNumber += str[pos];

m++;

State = EnumState.l2;

}

else

{

SetError(Err.DigitSpaceClosebracket, pos - 1);

State = EnumState.Error;

}

if ((State == EnumState.k0) || (State == EnumState.a19))

{

if (Int16.TryParse(IdNumber.Substring(0), out Int16 val))

{

if(!list\_of\_const.Contains(" - индекс - целая"))

{

index\_1 = IdNumber;

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - индекс - целая");

const\_count++;

IdNumber = "";

}

else

{

if (IdNumber != index\_1 && IdNumber != index)

{

IdNumber = const\_count.ToString() + ") " + IdNumber;

list\_of\_const.AddLast(IdNumber);

list\_of\_const.AddLast(" - индекс - целая");

const\_count++;

IdNumber = "";

}

}

IdNumber = "";

}

else

{

SetError(Err.ValueUotOfRange, pos - 1);

State = EnumState.Error;

}

}

break;

}

// ожидается пробел или закрывающая скобка

case EnumState.a19:

{

if (str[pos] == ' ')

State = EnumState.a19;

else if (str[pos] == ']')

State = EnumState.k0;

else

{

SetError(Err.SpaceClosebracket, pos);

State = EnumState.Error;

}

break;

}

// ожидаем пробел, мат операции или точку с запятой

case EnumState.k0:

{

if (str[pos] == ' ')

State = EnumState.x1;

else if ((str[pos] == '+') || (str[pos] == '-') || (str[pos] == '\*') || (str[pos] == '/'))

State = EnumState.a16;

else if (str[pos] == ';')

State = EnumState.Final;

else

{

SetError(Err.SpaceSemicolonMathOperExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается мат операция, пробел, D, M, точка с запятой

case EnumState.x1:

{

if (str[pos] == ' ')

State = EnumState.x1;

else if ((str[pos] == '+') || (str[pos] == '-') || (str[pos] == '\*') || (str[pos] == '/'))

State = EnumState.a16;

else if (str[pos] == ';')

State = EnumState.Final;

else if (str[pos] == 'M')

State = EnumState.x2;

else if (str[pos] == 'D')

State = EnumState.y2;

else

{

SetError(Err.LetterMathOperSpaceOpenbracketSemicolon, pos);

State = EnumState.Error;

}

break;

}

// ожидается опреденная буква

case EnumState.x2:

{

if (str[pos] == 'O')

State = EnumState.x3;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается опреденная буква

case EnumState.x3:

{

if (str[pos] == 'D')

State = EnumState.x4;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается опреденная буква

case EnumState.y2:

{

if (str[pos] == 'I')

State = EnumState.y3;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается опреденная буква

case EnumState.y3:

{

if (str[pos] == 'V')

State = EnumState.x4;

else

{

SetError(Err.LetterExpected, pos);

State = EnumState.Error;

}

break;

}

// ожидается пробел

case EnumState.x4:

{

if (str[pos] == ' ')

State = EnumState.a16;

else

{

SetError(Err.SpaceExpected, pos);

State = EnumState.Error;

}

break;

}

default:

{

State = EnumState.Error;

break;

}

}

}

pos++;

}

return (State == EnumState.Final);

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab\_taifya

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

int currentPosition = input\_box.SelectionStart;

input\_box.SelectAll();

//input\_box.SelectionFont = new Font(DefaultFont, FontStyle.Regular);

input\_box.SelectionColor = Color.Black;

identificators\_and\_constants.Items.Clear();

var text = input\_box.Text.ToUpper();

ClassResult a = ClassCheck.Check(text);

label3.Text = a.ErrMessage;

if (a.ErrPosition == -1)

{

identificators\_and\_constants.Items.Add("СПИСОК ИДЕНТИФИКАТОРОВ:");

identificators\_and\_constants.Items.Add(" Имя - Вид");

a.ListOfIds.ToList().ForEach((value) => identificators\_and\_constants.Items.Add(value));

identificators\_and\_constants.Items.Add("СПИСОК КОНСТАНТ:");

identificators\_and\_constants.Items.Add(" Значение - Вид - Тип");

a.ListOfConst.ToList().ForEach((value) => identificators\_and\_constants.Items.Add(value));

}

else

{

label3.ForeColor = Color.Red;

label3.Text = a.ErrMessage;

input\_box.Select(a.ErrPosition, input\_box.Text.Length - a.ErrPosition);

if (input\_box.SelectionLength > 0)

input\_box.SelectionLength = 1;

input\_box.Focus();

}

}

private void button1\_Click\_1(object sender, EventArgs e)

{

input\_box.Clear();

identificators\_and\_constants.Items.Clear();

}

}

}