**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«Московский энергетический институт»**

**Кафедра математического и компьютерного моделирования**

**«Математическое обеспечение ЭВМ»**

Лабораторная работа №3

**"Построение синтаксического анализатора"**

Вариант №11

Выполнил: Симаков А.М.

Группа: А-16-20

Преподаватель: Князев А.В.

**Москва 2021г**

1. **Задание.**

*Общее:*

Преобразовать заданную грамматику в LL(1)-грамматику.

2. Разработать МП-автомат для нисходящего грамматического разбора

предложений данного языка.

3. Разработать функцию, реализующую МП-автомат.

4. Разработать программу, иллюстрирующую работу созданного автомата

для данного языка. Программа разрабатывается как приложение с

графическим интерфейсом на языке C# в среде Visual Studio. Не должны

использоваться коллекции. Не должны использоваться регулярные

выражения и другие средства разбора строк.

*Индивидуальное:*

Оператор присваивания:

<ид.>:=<ар.выр.>;

Условный оператор:

if(<лог.выр.>) then <совок. операторов> [else < совок.

операторов>] endif

Оператор цикла:

ДЛЯ <ид.>:=<F> ДО <F> [ШАГ <F>] ЦИКЛ <совок. опе-

раторов> КОНЕЦ

Арифметическое выражение:

<E>::=<T><E-список>

<E-список>::= +<T><E-список>

<E-список>::= ε

<T>::=<F><T-список>

<T-список>::= \*<F><T-список>

<T-список>::= ε

<F>::=<Id>

<F>::=<Int>

Логическое выражение:

<лог.выр.>::=<F><лог.опер.><F>

<лог.опер.>::= =

<лог.опер.>::= !=

Пример программы:

a:=16\*3+1;

b:=11+2\*a;

c:=3\*a+2;

if(b!=c) then a:=4\*b; b:=12; else a:=2\*b+3; endif

k:=0; s:=0;

ДЛЯ i:=1 ДО 10 ШАГ 2 ЦИКЛ

k:=k+1;

s:=s+k;

КОНЕЦ

1. **LL(1) – грамматика**

<Program> ::= <S>  
<Program> ::= ɛ  
<S>::=<Id>:=<E>;<Program>  
<S>::= if (<Log>)then<S><SList>endif <Program>  
<S>::= ДЛЯ <id>:=<F>ДО<F><Step>ЦИКЛ<S> КОНЕЦ

<Id>:=I  
<E>::=<T><EList>  
<EList >::= +<T><EList>  
<EList >::= ε  
<T>::=<F><TList>  
<TList>::= \*<F><TList>  
<TList>::= ε  
<F>::= I  
<F>::= C  
<SList>:= else<S>  
<SList>:= ε  
<Log>.::=<F><LopOp>  
<LogOp>::= =<F>  
<LogOp>::= !=<F>

<Step>::=ШАГ<F>

<Step>::= ε

1. **Множества выбора**

Выбор(1) = Перв(<S>) = {I, if, ДЛЯ}  
Выбор(2) = След(<Program>) = { else, endif, КОНЕЦ }  
Выбор(3) = Перв(<Id>:=<E>;<Program>) = {I}  
Выбор(4) = Перв (if (<Log>)then<S><SList><Program>) = { ЕСЛИ }  
Выбор(5) = Перв(ДЛЯ <id>:=<F>ДО<F><Step>ЦИКЛ<S>КОНЕЦ) = {ДЛЯ}  
Выбор(6) = Перв(I) = {I}  
Выбор(7) = Перв (<T><EList>) = {I, C, (}  
Выбор(8) = Перв (+<T><EList>) = {+}  
Выбор(9) = След(<EList>) = {), ;}  
Выбор(10) = Перв(<F><TList>) = {Id, C, (}  
Выбор(11) = Перв(\*<F><TList>) = {\*}  
Выбор(12) = След (<TList>) = {+, ), ;}  
Выбор(13) = Перв (C) = {C}  
Выбор(14) = Перв (else<S>) = {else}  
Выбор(15) = След(<SList>) = {I, if, ДЛЯ}  
Выбор(16) = Перв (<F><LogOp>) = {I, C}  
Выбор(17) = Перв (=<F>) = {=}  
Выбор(18) = Перв (!=<F>) = {!=}

Выбор(19) = Перв (ШАГ<F>) = {ШАГ}  
Выбор(20) = След (<Step>) = {ЦИКЛ}

1. **Управляющая таблица LL(1) – грамматики**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | I | C | + | \* | ( | ) | ; | := | = | != | if | then | else | endif | ДЛЯ | ДО | ШАГ | ЦИКЛ | КОНЕЦ | ⊥ |
| ­<S> | #1 | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | #2 | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | #3 | Отвергнуть | Отвергнуть |
| <Program> | #4 | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | #4 | Отвергнуть | Вытолкнуть, Держать | Вытолкнуть, Держать | Отвергнуть | Отвергнуть | Отвергнуть | #4 | Вытолкнуть, Держать | Отвергнуть |
| <Id> | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <E> | #5 | #5 | Отвергнуть | Отвергнуть | #5 | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <T> | #7 | #7 | Отвергнуть | Отвергнуть | #7 | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <EList> | Отвергнуть | Отвергнуть | #6 | Отвергнуть | Отвергнуть | Вытолкнуть, Держать | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <TList> | Отвергнуть | Отвергнуть | Вытолкнуть, Держать | #8 | Отвергнуть | Вытолкнуть, Держать | Вытолкнуть, Держать | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <F> | Вытолкнуть, Сдвиг | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <SList> | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | #9 | Вытолкнуть, Держать | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <Log> | #10 | #10 | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <LogOp> | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | #11 | #12 | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| <Step> | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | #13 | Вытолкнуть, Держать | Отвергнуть | Отвергнуть |
| := | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| ДО | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| ТО | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| ( | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| ) | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| КОНЕЦ\_ЕСЛИ | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Вытолкнуть, Сдвиг | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть |
| КОНЕЦ\_ЦИКЛ | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Вытолкнуть, Сдвиг | Отвергнуть |
| ∇ | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Отвергнуть | Допустить |

Начальное содержимое магазина: <Program>∇

Правила замены

#1 Заменить(<Id>:=<E>;<Program>), Держать  
#2 Заменить((<Log>)then<S><SList><Program>), Сдвиг  
#3 Заменить(<Id> := <F> ДО <F><Step>ЦИКЛ<S> КОНЕЦ), Сдвиг  
#4 Заменить(<S>), Держать  
#5 Заменить(<T><EList>), Держать  
#6 Заменить(<T><EList>), Сдвиг  
#7 Заменить(<F><TList>), Держать

#8 Заменить(<F><TList>), Держать  
#9 Заменить(<S>), Сдвиг  
#10 Заменить(<F><LogOp>), Держать

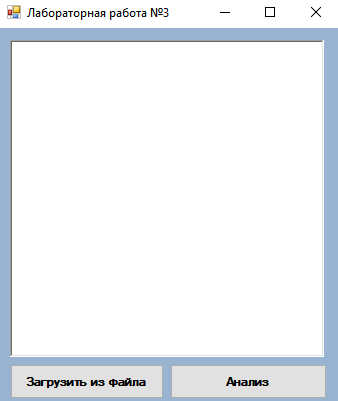
#11 Заменить(<F>),Сдвиг

#12 Заменить(<F>),Сдвиг

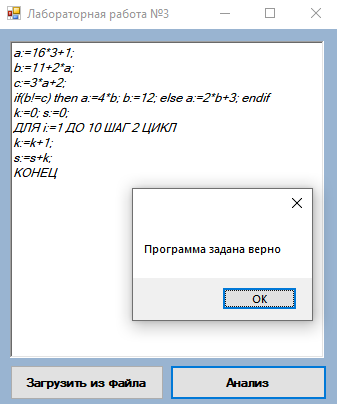
#13 Заменить(<F>),Сдвиг

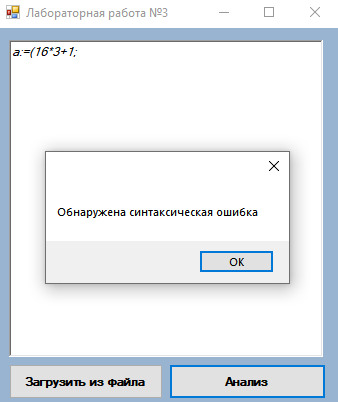
1. **Описание работы программы**

Выводится форма с одним текстовым полем.



Вводим код в текстовое поле и нажимаем «Анализ». Код можно  
загрузить из файла. После этого выскакивает окошко с сообщением: если программа задана верно, то будет написано «Программа задана верно», если найдены ошибки, то об этом также выведется сообщение.

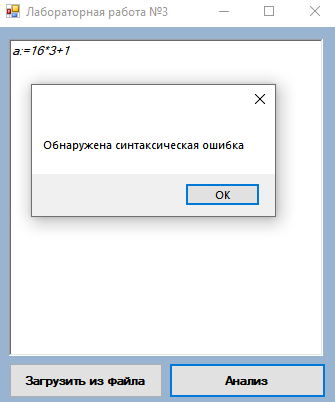


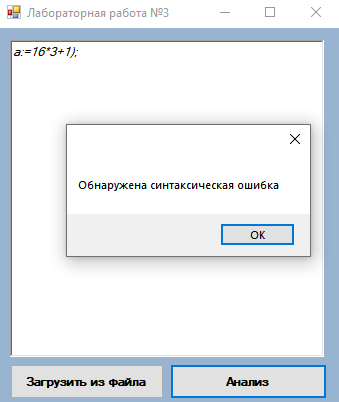


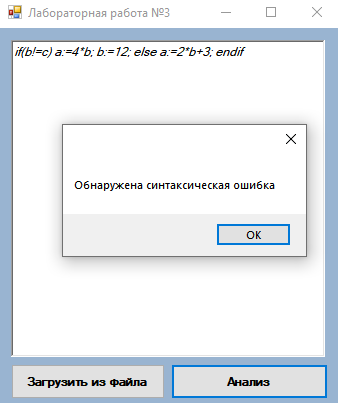
1. **Алгоритмы работы синтаксического анализатора**  
   Кладем стек начальный символ <Program>.  
   Заводим переменную для сохранения действий держания лексемы, сдвига или конца работы синтаксического анализатора (action).  
   Просим у лексического анализатора первую лексему.  
   Пока лексический блок не прочтет всю строку, либо пока не найдена ошибка  
    смотрим последний символ стека  
    определяем класс текущей лексемы  
    исходя из последнего символа стека, вызываем метод (для   
    каждого символа он свой), определяющий на основе класса   
    лексемы правило замены в стеке и возвращающий действие для   
    action – сдвиг, задержка или ошибка  
    если в стеке находился терминальный символ, то в action  
    передаем сдвиг  
    если обнаружена ошибка – прервать программу  
    если action равно «сдвиг», то берем у лексического анализатора  
    следующую лексему  
    если лексический анализатор прочел всю строку  
    то если в стеке лежит <Program>, то допустить строку, закончить

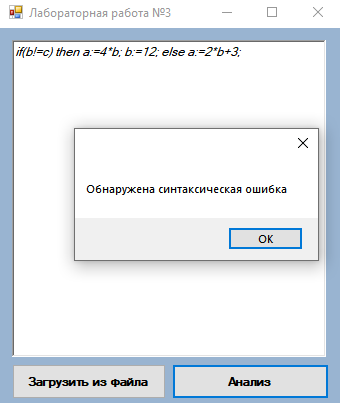
анализ

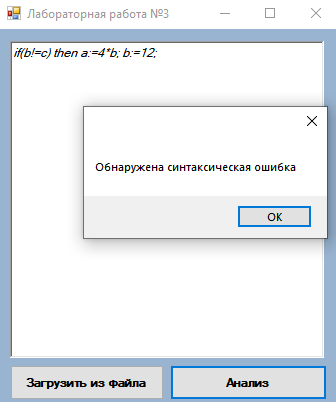
1. **Тесты**

Ошибки в присвоении  


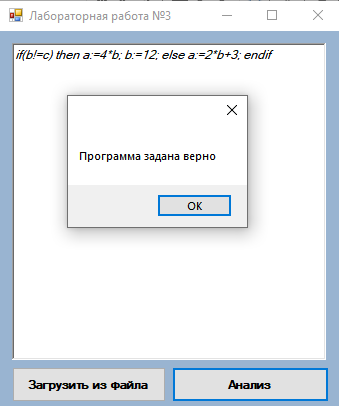


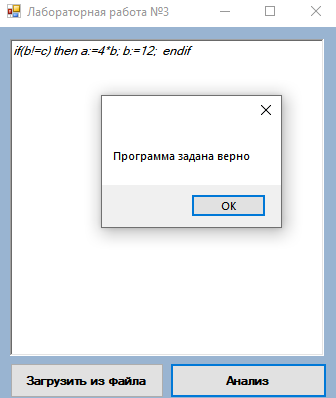
Ошибки в условном операторе  
 

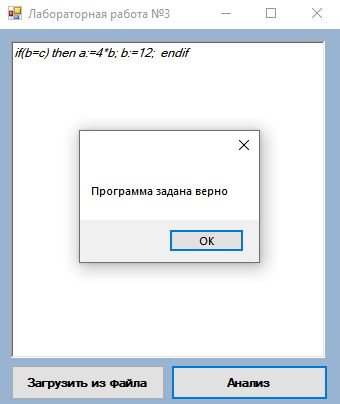


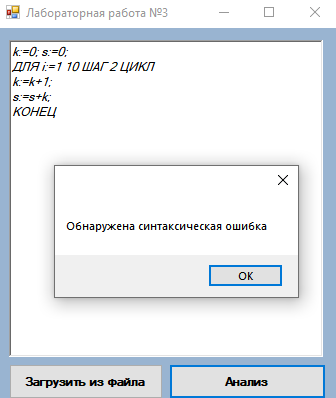


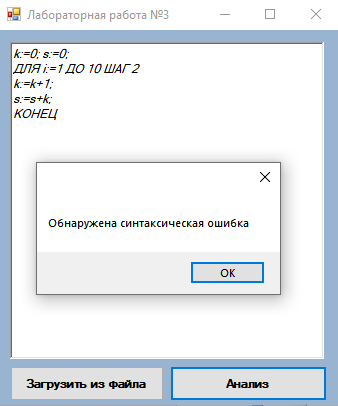
Правильная работа условного оператора

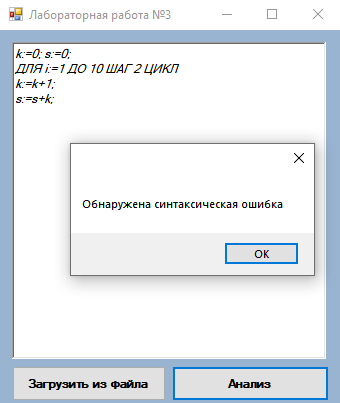




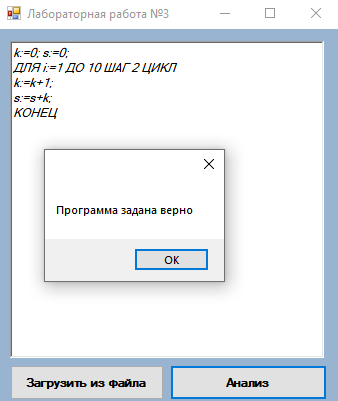


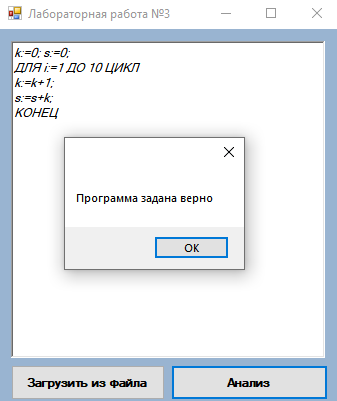
Ошибки в цикле  




****

Правильная работа программы с циклом





1. **Листинг программы**

using System;

using System.IO;

using System.Windows.Forms;

namespace Lab3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

richTextBox1.Clear();

if (File.Exists("txt1.txt"))

{

string[] readText = File.ReadAllLines("txt1.txt");

for (int i = 0; i < readText.Length - 1; i++)

{

richTextBox1.AppendText(readText[i] + "\n");

}

richTextBox1.AppendText(readText[readText.Length - 1]);

}

else

{

MessageBox.Show("Не удалось открыть файл", "Файл не найден", MessageBoxButtons.OK);

}

}

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

{

string pro = richTextBox1.Text;

Lexer lexer = new Lexer(pro);

Lexem lexem = lexer.GetLex();

while (lexem != null)

{

if (lexem.GetLexemClass() == "error")

{

MessageBox.Show("Обнаружена лексическая ошибка");

return;

}

lexem = lexer.GetLex();

}

lexer = new Lexer(pro);

SyntaxAnalysis synt = new SyntaxAnalysis();

if (synt.Analysis(lexer))

{

MessageBox.Show("Программа задана верно");

}

else

{

MessageBox.Show("Обнаружена синтаксическая ошибка");

return;

}

lexer = new Lexer(pro);

lexem = lexer.GetLex();

}

}

}

using System;

namespace Lab3

{

class Lexer

{

private int position = 0;

private string text;

private string buffer = "";

private int state = -1;

public Lexer(string str)

{

text = str;

}

private bool control(char ch)

{

if (ch == ' ' || ch == '\n' || ch == '\t')

return true;

return false;

}

private bool IsKeyword(string str)

{

if (String.Compare(str, "if") == 0 || String.Compare(str, "then") == 0 || String.Compare(str, "else") == 0 || String.Compare(str, "endif") == 0 || String.Compare(str, "ДЛЯ") == 0 || String.Compare(str, "ДО") == 0 || String.Compare(str, "ШАГ") == 0 || String.Compare(str, "ЦИКЛ") == 0 || String.Compare(str, "КОНЕЦ") == 0)

return true;

return false;

}

private bool IsIdentifier(char ch)

{

if (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z' || ch >= 'а' && ch <= 'я' || ch >= 'А' && ch <= 'Я')

return true;

return false;

}

private bool IsInteger(char ch)

{

if (ch >= '0' && ch <= '9')

return true;

return false;

}

private bool IsSeperate(char ch)

{

if (ch == '=' || ch == ';' || ch == '+' || ch == '\*' || ch == '!' || ch == ':' || ch == '(' || ch == ')')

return true;

return false;

}

private int keywords(string str)

{

switch (str)

{

case "if":

return 3;

case "then":

return 4;

case "else":

return 5;

case "endif":

return 6;

case "ДЛЯ":

return 7;

case "ДО":

return 8;

case "ШАГ":

return 9;

case "ЦИКЛ":

return 10;

case "КОНЕЦ":

return 11;

}

return -1;

}

private int seperates(char ch)

{

switch (ch)

{

case '=':

return 14;

case ';':

return 15;

case '+':

return 16;

case '\*':

return 17;

case '(':

return 18;

case ')':

return 19;

}

return -1;

}

private int ConvertState(int state, char ch)

{

if (state == 3)

{

if (ch == ':')

{

position += 1;

ch = text[position];

if (ch == '=')

return 12;

else

return -1;

}

else

{

if (ch == '!')

{

position += 1;

ch = text[position];

if (ch == '=')

return 13;

else

return -1;

}

else

return seperates(ch);

}

}

return state;

}

private int LexAnalysis(char ch)

{

if (control(ch))

return -1;

if (IsSeperate(ch))

return 3;

if (IsIdentifier(ch))

return 1;

if (IsInteger(ch))

return 2;

return -2;

}

public Lexem GetLex()

{

if (position >= text.Length)

return null;

char ch;

int analysis;

while (state == -1)

{

if (position >= text.Length)

return null;

ch = text[position];

analysis = LexAnalysis(ch);

if (analysis == -2)

{

Lexem lexem0 = new Lexem("\0", "error");

return lexem0;

}

if (analysis != -1)

{

if (analysis == 1)

{

state = 1;

buffer = Convert.ToString(ch);

}

else if (analysis == 2)

{

state = 2;

buffer = Convert.ToString(ch);

}

else state = 3;

}

else

position += 1;

}

ch = text[position];

if (state == 3)

{

string conv = Convert.ToString(ConvertState(state, ch));

Lexem lexem1;

if (conv == "12")

{

lexem1 = new Lexem(":=", conv);

state = -1;

position += 1;

return lexem1;

}

if (conv == "13")

{

lexem1 = new Lexem("!=", conv);

state = -1;

position += 1;

return lexem1;

}

Lexem lexem0 = new Lexem(Convert.ToString(ch), Convert.ToString(ConvertState(state, ch)));

state = -1;

position += 1;

return lexem0;

}

position += 1;

if (position >= text.Length - 1)

{

Lexem lexem0 = new Lexem(buffer, Convert.ToString(ConvertState(state, ch)));

position += 1;

state = -1;

return lexem0;

}

ch = text[position];

analysis = LexAnalysis(ch);

while (analysis != -1 && analysis < 3 && position < text.Length)

{

if (state == 2 && analysis == 1 || analysis == -2)

{

Lexem lexem0 = new Lexem("\0", "error");

return lexem0;

}

buffer = String.Concat(buffer, ch);

position += 1;

if (position == text.Length)

break;

ch = text[position];

analysis = LexAnalysis(ch);

}

if (IsKeyword(buffer))

{

Lexem lexem0 = new Lexem(buffer, Convert.ToString(keywords(buffer)));

state = -1;

return lexem0;

}

Lexem lexem = new Lexem(buffer, Convert.ToString(ConvertState(state, ch)));

state = -1;

return lexem;

}

}

class Lexem

{

private string lexem;

private string lexem\_class;

public Lexem(string lexem, string lexem\_class)

{

this.lexem = lexem;

this.lexem\_class = lexem\_class;

}

public string GetLexem()

{

return lexem;

}

public string GetLexemClass()

{

return lexem\_class;

}

}

}

namespace Lab3

{

class SyntaxAnalysis

{

Stack<string> stack = new Stack<string>();

public bool Analysis(Lexer lexer)

{

stack.Push("Program");

int action;

string LexClass;

string state;

Lexem lexem = lexer.GetLex();

if (lexem == null)

{

return true;

}

for (; ; )

{

if (stack.IsEmpty())

return false;

state = stack.Top();

LexClass = lexem.GetLexemClass();

switch (state)

{

case "S":

action = S(LexClass);

break;

case "Program":

action = K(LexClass);

break;

case "Id":

action = Id(LexClass);

break;

case "E":

action = E(LexClass);

break;

case "EList":

action = EList(LexClass);

break;

case "T":

action = T(LexClass);

break;

case "TList":

action = TList(LexClass);

break;

case "F":

action = F(LexClass);

break;

case "SList":

action = SList(LexClass);

break;

case "Log":

action = Log(LexClass);

break;

case "LogOp":

action = LogOp(LexClass);

break;

case "Step":

action = Step(LexClass);

break;

default:

if (state == lexem.GetLexem())

{

stack.Pop();

action = 2;

}

else

action = -1;

break;

}

if (action == -1)

return false;

if (action == 2)

{

lexem = lexer.GetLex();

if (lexem == null)

if (stack.Top() == "Program" && stack.Length() == 1)

{

return true;

}

else return false;

}

}

}

private int S(string lclass)

{

switch (lclass)

{

//#1

case "1":

stack.Pop();

stack.Push("Program");

stack.Push(";");

stack.Push("E");

stack.Push(":=");

stack.Push("Id");

return 1;

//#2

case "3":

stack.Pop();

stack.Push("Program");

stack.Push("endif");

stack.Push("SList");

stack.Push("S");

stack.Push("then");

stack.Push(")");

stack.Push("Log");

stack.Push("(");

return 2;

//#3

case "7":

stack.Pop();

stack.Push("Program");

stack.Push("КОНЕЦ");

stack.Push("S");

stack.Push("ЦИКЛ");

stack.Push("Step");

stack.Push("F");

stack.Push("ДО");

stack.Push("F");

stack.Push(":=");

stack.Push("Id");

return 2;

default:

return -1;

}

}

private int K(string lclass)

{

//#4

if (lclass == "1" || lclass == "3" || lclass == "7")

{

stack.Pop();

stack.Push("S");

return 1;

}

//#5

if (lclass == "5" || lclass == "6" || lclass == "11")

{

stack.Pop();

return 1;

}

return -1;

}

private int Id(string lclass)

{

//#6

if (lclass == "1")

{

stack.Pop();

return 2;

}

return -1;

}

private int E(string lclass)

{

//#7

if (lclass == "1" || lclass == "2" || lclass == "18")

{

stack.Pop();

stack.Push("EList");

stack.Push("T");

return 1;

}

return -1;

}

private int EList(string lclass)

{

//#8

if (lclass == "16")

{

stack.Pop();

stack.Push("EList");

stack.Push("T");

return 2;

}

//#5

if (lclass == "19")

{

stack.Pop();

return 1;

}

//#6

if (lclass == "15")

{

stack.Pop();

return 1;

}

return -1;

}

private int T(string lclass)

{

//#9

if (lclass == "1" || lclass == "2" || lclass == "18")

{

stack.Pop();

stack.Push("TList");

stack.Push("F");

return 1;

}

return -1;

}

private int TList(string lclass)

{

//#10

if (lclass == "17")

{

stack.Pop();

stack.Push("TList");

stack.Push("F");

return 2;

}

//#5

if (lclass == "16" || lclass == "15" || lclass == "19")

{

stack.Pop();

return 1;

}

return -1;

}

private int F(string lclass)

{

//#6

if (lclass == "1" || lclass == "2")

{

stack.Pop();

return 2;

}

return -1;

}

private int Step(string lclass)

{

//#6

if (lclass == "9")

{

stack.Pop();

stack.Push("F");

return 2;

}

if (lclass == "10")

{

stack.Pop();

return 1;

}

return -1;

}

private int SList(string lclass)

{

if (lclass == "5")

{

stack.Pop();

stack.Push("Program");

stack.Push("S");

return 2;

}

if (lclass == "6")

{

stack.Pop();

return 1;

}

return -1;

}

private int Log(string lclass)

{

//#12

if (lclass == "1" || lclass == "2")

{

stack.Pop();

stack.Push("LogOp");

stack.Push("F");

return 1;

}

return -1;

}

private int LogOp(string lclass)

{

//#6

if (lclass == "14")

{

stack.Pop();

stack.Push("F");

return 2;

}

if (lclass == "13")

{

stack.Pop();

stack.Push("F");

return 2;

}

return -1;

}

}

}

namespace Lab3

{

class Stack<T>

{

private int head { get; set; }

private T[] mass = new T[100];

public Stack()

{

head = 0;

}

public bool IsEmpty()

{

return head == 0;

}

public void Push(T data)

{

mass[head] = data;

head++;

}

public T Pop()

{

mass[head] = default;

head--;

return mass[head];

}

public T Top()

{

return mass[head - 1];

}

public int Length()

{

return head;

}

public T Ind(int a)

{

return mass[a];

}

public void Delete(int a)

{

for (int i = a; i < head; i++)

{

mass[i] = mass[i + 1];

}

head--;

}

}

}