МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА»

(САМАРСКИЙ УНИВЕРСИТЕТ)

Отчёт по лабораторной работе

по курсу «Теория формальных языков и грамматик»

Вариант №16.

Выполнил:

Никитин А.А.

гр.6303

Проверил: Литвинов В. Г.

Самара 2016

**Задание:**

Написать программу синтаксического анализа автоматного языка вызова процедур языка Fortran - 77; цепочки языка имеют вид:

**CALL** <идентификатор>[**(**<список фактических параметров>**)**]

<список фактических параметров> :: = <параметр>[**,**<список фактических параметров>]

⎧<константа любая>

<параметр> :: = ⎨ ⎡**⎛**⎧<константа целая>⎫**⎞**⎤

⎪<идентификатор> ⎢**⎜**⎨ ⎬**⎟**⎥

⎩ ⎣**⎝**⎩<идентификатор> ⎭**⎠**⎦

<идентификатор> - идентификатор, начинается с буквы, включает последовательность букв, цифр, не допускает пробелы и специальные символы, ввести ограничение на длину (не более 8 символов) и не может быть зарезервированным словом (CALL, FORMAT, FOR, TO);

<константа целая> - целое число в диапазоне 1 ÷ +32767;

<константа любая> - целое число в диапазоне -32768 ÷ +32767, число с фиксированной точкой, число с плавающей точкой. Число цифр порядка ≤ 2.

Семантика:

Построить таблицу идентификаторов и констант с указанием номера параметра. Учесть перечисленные выше ограничения на идентификаторы и константы. Не допускать дублирование идентификаторов.

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

Примеры правильных цепочек:

CALL SINUS

CALL PRIMER1 ( ABC, 3.14, 0.25E4, 15 )

CALL PRIMER2 ( A ( B ), C ( 35 ), 34E-2, AB ( KB ) )

**Детерминированный конечный автомат языка:**

**Листинг программы:**

# Класс Result

using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;

namespace Analizator

{

public enum Err {

NoError,

UnknownError,

OverflowCharacters,

OverflowTransfer,

OutOfRange,

FormatExpected,

XExpected,

MathematicalErrors,

ExpectedBrace1,

ExpectedBrace2,

ExpectedListElement,

ExpectedElement,

ExpectedPoint,

ConstExpected,

ElementExpected,

Error1

}

class Result { int ErrPos; Err Err; string \_Str;

public Result(int ErrPos, Err Err, string Value)

{

this.ErrPos = ErrPos; this.Err = Err;

\_Str = Value;

}

public int ErrPosition

{ get {

return ErrPos;

}

} public string ErrMessage

{ get { switch (Err) { case Err.NoError: {

return "Нет ошибок";

} case Err.UnknownError: {

return "Неизвестная ошибка";

} case Err.OutOfRange:

{

return "Выход за границы анализируемой строки";

} case Err.OverflowCharacters:

{

return "Входных символов больше 50";

} case Err.Error1:

{

return "Слишком большое число";

} case Err.OverflowTransfer:

{

return "Превышено количество знаков перевода строки";

} case Err.FormatExpected:

{

return "Ожидается ключевое слово \"FORMAT\"";

} case Err.ConstExpected:

{

return "Ожидается константа";

} case Err.ExpectedBrace1:

{

return "Ожидается открывающаяся скобка";

} case Err.ExpectedBrace2:

{

return "Ожидается закрывающаяся скобка";

} case Err.ExpectedElement:

{

return "Ожидается элемент";

} case Err.ExpectedListElement:

{

return "Ожидается список элементов";

} case Err.ElementExpected:

{

return "Ожидается элемент";

} case Err.ExpectedPoint:

{

return "Ожидается точка";

} case Err.XExpected:

{

return "Ожидается \"X\"";

} case Err.MathematicalErrors:

{

return "Константа1 больше чем Константа2 + 2";

} default: {

return "Неизвестная ошибка";

}

}

}

}

public string Str

{ get {

return \_Str;

}

}

}

}

# Класс Analizator

using System;

using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;

namespace Analizator

{

static class Analizator

{

enum State { Start, Error, Final, A, B, C, D, L, G, H };

private static int i = 0;

private static int max = 50;

private static string st;

private static int len;

private static Err Err;

private static int ErrPos;

private static string \_const;

private static int Const1;

private static int Const2;

private static int Transfer = 0;

private static string str;

public static Result Result(string value)

{ str = ""; st = value; i = 0; len = st.Length; Transfer = 0;

Const2 = 0;

Const1 = 0;

\_const = null;

SetError(Err.NoError, -1);

Analiz();

return new Result(ErrPos, Err, str);

}

private static void SetError(Err ErrorType, int ErrorPosition)

{

Err = ErrorType;

ErrPos = ErrorPosition;

}

private static bool Letter()

{

if (char.IsLetter(st[i]))

{ i++; return true;

} else {

return false;

}

}

private static bool Digit()

{

if (char.IsDigit(st[i]))

{ i++; return true;

} else {

return false;

}

}

private static bool EqualSymbol(char Symbol)

{

if (i >= len) {

return false;

}

if (Symbol == st[i])

{ i++; return true;

} else {

return false;

}

} private static bool Format() {

State Sta = State.Start; int TmpPos = i;

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

{

if (i >= max)

{

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

} else if (i >= len) {

SetError(Err.OutOfRange, i - 1);

Sta = State.Error;

} else {

switch (Sta) { case State.Start: {

if (EqualSymbol('F'))

{

Sta = State.A;

} else

{

SetError(Err.FormatExpected, i);

Sta = State.Error;

} } break; case State.A: {

if (EqualSymbol('O'))

{

Sta = State.B;

} else

{

SetError(Err.FormatExpected, i);

Sta = State.Error;

} } break; case State.B: {

if (EqualSymbol('R'))

{

Sta = State.C;

} else

{

SetError(Err.FormatExpected, i);

Sta = State.Error;

} } break; case State.C: {

if (EqualSymbol('M'))

{

Sta = State.D;

} else

{

SetError(Err.FormatExpected, i);

Sta = State.Error;

} } break; case State.D: {

if (EqualSymbol('A'))

{

Sta = State.G;

} else

{

SetError(Err.FormatExpected, i); Sta = State.Error;

} } break; case State.G: {

if (EqualSymbol('T'))

{

Sta = State.Final;

} else

{

SetError(Err.FormatExpected, i);

Sta = State.Error;

}

} break; default:

{

SetError(Err.UnknownError, i);

Sta = State.Error;

} break;

}

} }

if (Sta == State.Error)

{

TmpPos = i; return false;

} else {

return true;

}

}

private static bool Constant()

{

State Sta = State.Start; int TmpPos = i; string c = "";

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

{

if (i >= 50)

{

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

}

else if (i >= len)

{

SetError(Err.OutOfRange, i - 1);

Sta = State.Error;

} else {

switch (Sta) { case State.Start: {

if (Digit()) {

c += st[i - 1];

Sta = State.B;

} else {

Sta = State.Error;

} } break; case State.B: {

if (Digit()) {

c += st[i - 1];

} else

{

Sta = State.Final;

} } break; default:

{

SetError(Err.UnknownError, i);

Sta = State.Error;

} break;

}

} }

if (Sta == State.Error)

{

TmpPos = i; return false;

} else

{

\_const = c; return true;

}

}

private static bool Element()

{

State Sta = State.Start; int TmpPos = i; Transfer = 0;

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

{

if (i >= max)

{

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

}

else if (i >= len)

{

SetError(Err.OutOfRange, i - 1);

Sta = State.Error;

} else {

switch (Sta) { case State.Start: {

if (EqualSymbol(' '))

{

Sta = State.Start;

}

else if (EqualSymbol('\"'))

{

Sta = State.B;

}

else if (EqualSymbol('F'))

{

Sta = State.A;

}

else if (EqualSymbol('I'))

{

Sta = State.D;

}

else if (EqualSymbol('/'))

{

Transfer++; if (Transfer > 3)

{

Sta = State.Error;

SetError(Err.OverflowTransfer, i - 1);

} else

{

Sta = State.L; str+="\n";

} }

else if (Constant())

{

Sta = State.G; int aa;

if (Int32.TryParse(\_const, out aa))

{

Sta = State.G;

} else

{

SetError(Err.Error1, i - 1);

Sta = State.Error;

} } else

{

SetError(Err.ElementExpected, i - 1);

Sta = State.Error;

} } break; case State.A: {

if (Constant())

{

Sta = State.C; int aa;

if (Int32.TryParse(\_const, out aa))

Const1 = aa; else

{

SetError(Err.Error1, i - 1);

Sta = State.Error;

} } else

{

SetError(Err.ConstExpected, i - 1);

Sta = State.Error;

} } break; case State.B: {

if (Letter() || Digit())

{

Sta = State.B; str += st[i - 1];

}

else if (EqualSymbol('\"'))

{

Sta = State.Final;

} else

{

Sta = State.Error;

} } break; case State.C: {

if (EqualSymbol('.'))

{

Sta = State.H;

} else

{

SetError(Err.ExpectedPoint, i - 1);

Sta = State.Error;

} } break; case State.D: {

if (Constant())

{ int aa;

if (Int32.TryParse(\_const, out aa))

{

Sta = State.Final; for (int j = 0; j < aa; j++)

{

if (j == 0) {

str += "F";

} else {

str += "I";

}

} } else

{

SetError(Err.Error1, i - 1);

Sta = State.Error;

} } else

{

SetError(Err.ConstExpected, i - 1);

Sta = State.Error;

} } break; case State.G: {

if (EqualSymbol('X'))

{

Sta = State.Final;

for (int j = 0; j < Convert.ToInt32(\_const); j++) {

str+= "\_";

} } else

{

SetError(Err.XExpected, i - 1);

Sta = State.Error;

} } break; case State.H: {

if (Constant())

{ int aa;

if (Int32.TryParse(\_const, out aa))

{

Const2 = aa;

if (Const1 < Const2 + 2)

{

SetError(Err.MathematicalErrors, i - 1);

Sta = State.Error;

} else

{

Sta = State.Final;

for (int j = 0; j < Convert.ToInt32(Const1); j++)

{

if (j == 0)

{

str += "F";

}

else if (j == Const1 - Const2 - 1)

{

str += ".";

} else {

str += "I";

}

}

} } else

{

SetError(Err.Error1, i - 1);

Sta = State.Error;

} } else

{

SetError(Err.ConstExpected, i - 1);

Sta = State.Error;

} } break; case State.L: {

if (EqualSymbol('/'))

{

Transfer++; if (Transfer > 3)

{

Sta = State.Error;

SetError(Err.OverflowTransfer, i - 1);

} else

{

Sta = State.L; str+="\n";

} }

else if (EqualSymbol(' '))

{

Sta = State.L;

} else

{

Sta = State.Final;

} } break; default:

{

Sta = State.Error;

} break;

}

} }

if (Sta == State.Error)

{

TmpPos = i; return false;

} else {

return true;

}

}

private static bool ListElement()

{

State Sta = State.Start; int TmpPos = i;

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

{

if (i >= max)

{

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

}

else if (i >= len)

{

SetError(Err.OutOfRange, i - 1);

Sta = State.Error;

} else {

switch (Sta) { case State.Start: {

if (EqualSymbol(' '))

{

Sta = State.Start;

}

else if (Element())

{

Sta = State.A;

} else {

Sta = State.Error;

} } break; case State.A: {

if (EqualSymbol(' '))

{

Sta = State.A;

}

else if (EqualSymbol(','))

{

Sta = State.Start;

} else

{

Sta = State.Final;

}

} break;

}

} }

if (Sta == State.Error)

{

TmpPos = i; return false;

} else {

return true;

}

}

private static bool Analiz() {

State Sta = State.Start; int TmpPos = i;

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

{

if (i >= max)

{

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

}

else if (i >= len)

{

SetError(Err.OutOfRange, i - 1);

Sta = State.Error;

} else {

switch (Sta) { case State.Start: {

if (Format())

{

Sta = State.A;

}

else if (EqualSymbol(' '))

{

Sta = State.Start;

} else { SetError(Err.FormatExpected, i - 1);

Sta = State.Error;

} } break; case State.A: {

if (EqualSymbol(' '))

{

Sta = State.A;

}

else if (EqualSymbol('('))

{

Sta = State.B;

} else

{

SetError(Err.ExpectedBrace1, i - 1);

Sta = State.Error;

} } break; case State.B: {

if (EqualSymbol(' '))

{

Sta = State.B;

}

else if (ListElement())

{

Sta = State.C;

} else

{

//SetError(Err.ExpectedListElement, i - 1);

Sta = State.Error;

} } break; case State.C: {

if (EqualSymbol(' '))

{

Sta = State.C;

}

else if (EqualSymbol(')'))

{

Sta = State.Final;

} else

{

SetError(Err.ExpectedBrace2, i - 1);

Sta = State.Error;

}

} break;

}

} }

if (Sta == State.Error)

{

TmpPos = i; return false;

} else {

return true;

}

}

}

}

# Класс пользовательского интерфейса

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 Analizator

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

label2.Text = ""; label3.Text = ""; string str = textBox1.Text; Result r = Analizator.Result(str); if (r.ErrPosition == -1)

{

richTextBox1.Text = r.Str;

} else {

string strs = r.ErrPosition.ToString() + "\n" + r.ErrMessage; richTextBox1.Text = strs;

}

}

private void Form1\_Load(object sender, EventArgs e) {

}

}

}

**Результат выполнения программы:**





