МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра автоматизированных систем управления



ОТЧЁТ

по РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЕ

**«***Комментарии языка FORTRAN***»**

по дисциплине: **«***Специализированное программное обеспечение***»**

Выполнил:Проверил:

Студент гр. «*АВТ-812*», «*АВТФ*» *«д.т.н., профессор»*

*Березин Дмитрий*

*«Шорников Юрий Владимирович»*

«\_\_\_» \_\_\_\_\_\_ 20\_\_г.«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2021

**Содержание**

[Введение 3](#_Toc85651924)

[1. Постановка задачи 4](#_Toc85651925)

[2. Разработка грамматики 5](#_Toc85651926)

[3. Грамматика для ANTLR 6](#_Toc85651927)

[4. Внедрение грамматики в интерфейс, созданной в ANTLR 7](#_Toc85651928)

[5. Тестовые примеры 9](#_Toc85651929)

[6. Листинг программы 11](#_Toc85651930)

[Заключение 12](#_Toc85651931)

[Список использованных источников 13](#_Toc85651932)

[Приложение А 14](#_Toc85651933)

Введение

Целью расчетно-графической работы является программная реализация сканера и парсера с помощью программы ANTLR и внедрение результата в интерфейс синтаксического анализатора для языка Fortran, разработанного в рамках курсовой работы по предмету «Теория формальных языков и компиляторов».

Расчетно-графическая работа содержит следующие разделы:

* Постановка задачи
* Используемая грамматика
* Грамматика для программы ANTLR
* Внедрение грамматики, созданной с помощью ANTLR
* Тестовые примеры
* Список использованных источников
* Листинг программы

1. Постановка задачи

Комментарий – это строка или несколько строк в исходном коде, которые могут прочесть разработчики, но они игнорируются компиляторами и интерпретаторами.

В языке Fortran реализованы комментарии, записываемые в одну строку. В строке, содержащей комментарий, первым символом всегда идет «\*» (звездочка).

Примеры комментариев на языке Fortran:

1. Строка «\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*» считается комментарием, так как начинается с символа «\*»
2. Строка «\*Foo = Bar» также является комментарием, так как начинается с символа «\*»
3. Разработка грамматики

Приступим к определению грамматики комментариев языка Fortran G[I].

В первую очередь необходимо определить язык, с помощью которого будут строиться комментарии:

<Буква> = {“a” | “b” | … | “z” | “A” | … | “Z”}

<Цифра> = {“0” | “1” | … | “9”}

<Разделитель> = {“.” | “,” | “!” | “?” | “(” | “)” | “{” | “}” | “[” | “]” | “+” | “-” | “=” | “\” | “/” | “<” | “>” | “ ”}

<Конец строки> = {“Перенос строки” | “Конец файла”}

На основе построенного языка комментариев определим правила вывода, с помощью которых будут формироваться комментарии:

1. <I> → “\*”<A> | (<Буква> | <Цифра> | <Разделитель>) <E> | <Конец строки>
2. <A> → (<Буква> | <Цифра> | <Разделитель> | “\*”) <A> | <Конец строки>
3. <E> →(<Буква>|<Цифра>|<Разделитель>| “\*”) <E> | <Конец строки>
4. Грамматика для ANTLR

Из разработанной грамматики построим грамматику для ANTLR. Каждое правило должно выглядеть следующим образом:

*rule: expr1 | expr2;*

Здесь в качестве *rule* принимается нетерминальный символ. *Expr1* может являться как терминальным, так и нетерминальным символом. *Expr2* – это альтернатива *expr1*.

Далее приведена реализация грамматики «Комментарии языка Fortran» для ANTLR:

grammar CommentsFortran;

prog: (comment WS)\* comment? EOF;

comment: STAR (LETTER|DIGIT|SPLITTER|STAR)\*;

STAR: '\*';

LETTER: ('a'..'z'|'A'..'Z');

DIGIT: ('0'..'9');

SPLITTER: ('.'|','|'!'|'?'|'('|')'|'{'|'}'|'['|']'|'+'|'-'|'='|'\\'|'/'|'<'|'>'|' ');

WS: ( '\r'? '\n');

1. Внедрение грамматики в интерфейс, созданной в ANTLR

В первую очередь для создания грамматики под язык C# необходимо добавить следующую опцию:

*options*

*{*

*language = CSharp;*

*}*

Далее, для подключения сгенерированных файлов в существующий проект с интерфейсом определим используемые в C# пространства имен:

*@parser::namespace { comment\_analyzer }*

*@lexer::namespace { comment\_analyzer }*

Далее генерируем исходный код лексера и парсера с помощью компилятора ANTLR. На выходе мы получаем следующий список файлов:

* CommentsFortran.interp
* CommentsFortran.tokens
* CommentsFortranBaseListener.cs
* CommentsFortranLexer.cs
* CommentsFortranLexer.interp
* CommentsFortranLexer.tokens
* CommentsFortranListener.cs
* CommentsFortranParser.cs

Полученные файлы копируем в папку с проектом интерфейса анализатора.

Далее в функции, выполняющей анализ исходного текста, необходимо подключить библиотеку *Antlr4.Runtime* и добавить следующие строки, чтобы начать использование сгенерированного в ANTLR кода:

// Выделяем элемент формы с исходным текстом

String source = StaticData.mainForm.TextBox.Text;

// Устанавливаем входной поток символов из элемента формы

ICharStream input = CharStreams.fromString(source);

// Настраиваем лексер на этот поток

CommentsFortranLexer lexer = new CommentsFortranLexer(input);

// Создаем поток токенов на основе лексера

CommonTokenStream tokens = new CommonTokenStream(lexer);

// Создаем парсер

CommentsFortranParser parser = new CommentsFortranParser(tokens);

// Удаляем стандартный обработчик ошибок

parser.RemoveErrorListeners();

// Объявляем самописный обработчик ошибок

CommentAnalyzerErrorListener customListener = new CommentAnalyzerErrorListener();

// Добавляем его в парсер

parser.AddErrorListener(customListener);

// Запускаем первое правило грамматики

parser.prog();

StaticData.mainForm.ResultsTextBox.Text = "";

// Далее идет вывод результатов анализа исходного текста

if (customListener.errorsList.Count == 0)

StaticData.mainForm.ResultsTextBox.Text = "No errors\r\n";

foreach (var error in customListener.errorsList)

{

StaticData.mainForm.ResultsTextBox.Text += error + "\r\n";

}

В этом блоке кода участвует также разработанный мной обработчик ошибок, позволяющий более точно оценить место ее появления. Исходный код обработчика будет приведен в Приложении А.

1. Тестовые примеры

На приведенных ниже рисунках 1 - 4 показаны примеры работы разработанной программы:

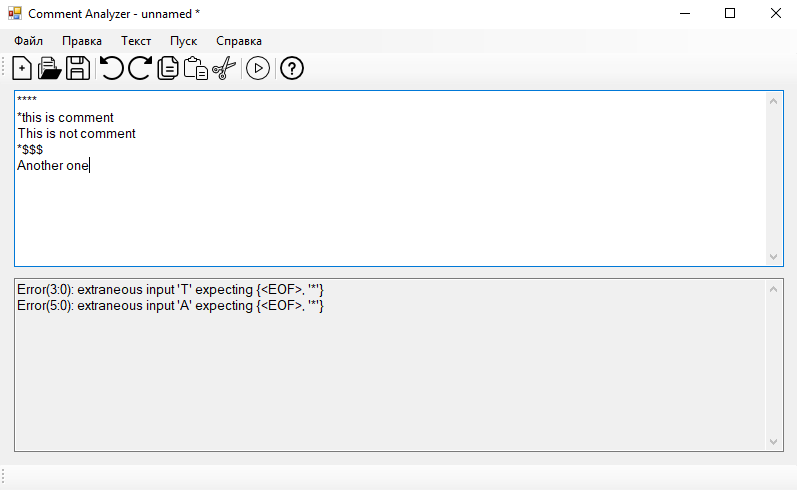


Рисунок 1 – Тестовый пример 1

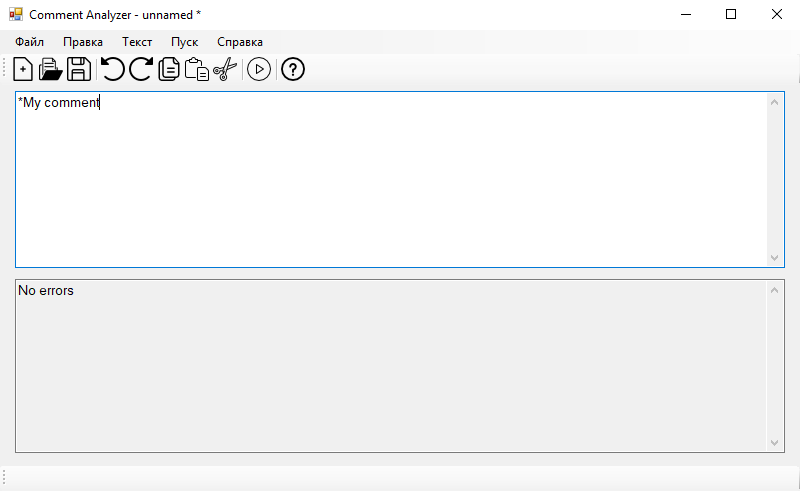


Рисунок 2 – Тестовый пример 2

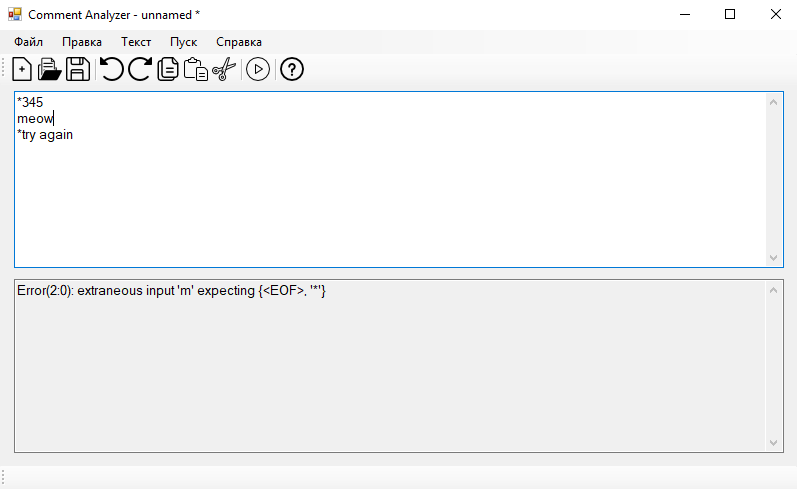


Рисунок 3 – Тестовый пример 3

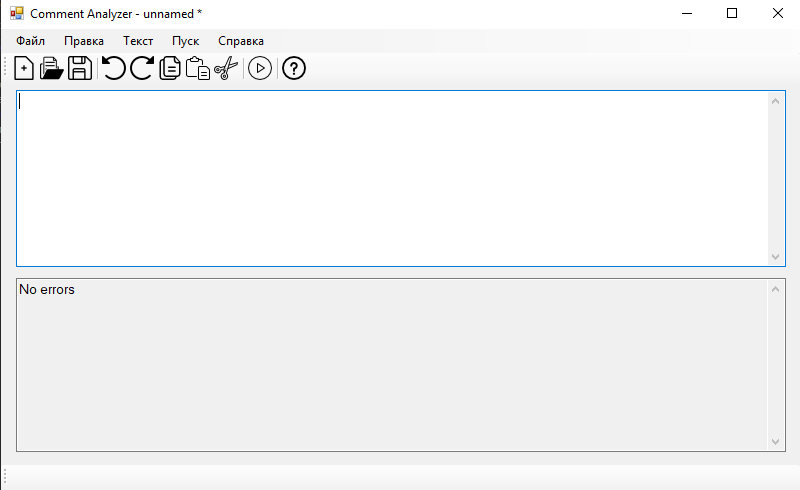


Рисунок 4 – Тестовый пример 4

1. Листинг программы

Листинг программной части сгенерированного с помощью ANTLR анализатора, а также реализация обработчика ошибок приведены в Приложении А.

Заключение

В результате выполнения расчетно-графической работы была реализована грамматика комментариев языка Fortran для ANTLR. Далее был получен модуль синтаксического анализатора на языке C# путем компиляции грамматики в программе ANTLR. Затем к ранее разработанному графическому интерфейсу на C# был подключен модуль анализатора с помощью библиотеки Antlr4.Runtime. Работа анализатора была протестирована на разных данных.

В качестве выводов по опыту работы с ANTLR могу сказать следующее: данная программа крайне удобна при проектировании синтаксических анализаторов, так как позволяет избежать ошибок реализации анализатора на целевом языке программирования. При ее использовании разработчик больше времени может уделить проектированию грамматики и поиску ошибок на этапе проектирования, а не ее реализации. В масштабных проектах это может оказывать существенное влияние. В качестве негативной стороны использования данной программы хочется привести ситуацию, при которой анализатор не в состоянии исправить незначительные ошибки или опечатки. Если при самостоятельной реализации анализатора можно добавить предупреждения об опечатках, исправить их и продолжить анализ исходного текста, то в реализации анализатора от ANTLR такой возможности нет.

Таким образом при реализации синтаксического анализатора на этапе проектирования следует определить, что является более приоритетной целью – время разработки или расширенная функциональность и гибкость конечного продукта в плане предоставления пользователю не только ошибок, но и предупреждений.

Список использованных источников

1. Компиляторы. Принципы, технологии, инструменты / Альфред Ахо, Рави Сети, Джеффри Ульман, Моника Лам, 2-e изд. М.: ООО "И.Д. Вильямс", 2008. 1184 с.
2. Теория и практика языковых процессоров / Ю.В. Шорников, Новосибирск: Изд-во НГТУ, 2004. 208 с.
3. Грамматика арифметики или пишем калькулятор на ANTLR // Хабр URL: https://habr.com/ru/post/110710/ (дата обращения: 05.10.2021).
4. First steps with ANTLR4 in C # // Franck Gaspoz URL: https://franckgaspoz.fr/en/first-steps-with-antlr4-in-csharp/ (дата обращения: 05.10.2021).

Приложение А

CommentsFortranLexer.cs

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated by a tool.

// ANTLR Version: 4.9.2

//

// Changes to this file may cause incorrect behavior and will be lost if

// the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

// Generated from CommentsFortran.g4 by ANTLR 4.9.2

// Unreachable code detected

#pragma warning disable 0162

// The variable '...' is assigned but its value is never used

#pragma warning disable 0219

// Missing XML comment for publicly visible type or member '...'

#pragma warning disable 1591

// Ambiguous reference in cref attribute

#pragma warning disable 419

using System;

using System.Collections;

using System;

using System.IO;

using System.Text;

using Antlr4.Runtime;

using Antlr4.Runtime.Atn;

using Antlr4.Runtime.Misc;

using DFA = Antlr4.Runtime.Dfa.DFA;

[System.CodeDom.Compiler.GeneratedCode("ANTLR", "4.9.2")]

[System.CLSCompliant(false)]

public partial class CommentsFortranLexer : Lexer {

protected static DFA[] decisionToDFA;

protected static PredictionContextCache sharedContextCache = new PredictionContextCache();

public const int

STAR=1, LETTER=2, DIGIT=3, SPLITTER=4, WS=5;

public static string[] channelNames = {

"DEFAULT\_TOKEN\_CHANNEL", "HIDDEN"

};

public static string[] modeNames = {

"DEFAULT\_MODE"

};

public static readonly string[] ruleNames = {

"STAR", "LETTER", "DIGIT", "SPLITTER", "WS"

};

public CommentsFortranLexer(ICharStream input)

: this(input, Console.Out, Console.Error) { }

public CommentsFortranLexer(ICharStream input, TextWriter output, TextWriter errorOutput)

: base(input, output, errorOutput)

{

Interpreter = new LexerATNSimulator(this, \_ATN, decisionToDFA, sharedContextCache);

}

private static readonly string[] \_LiteralNames = {

null, "'\*'"

};

private static readonly string[] \_SymbolicNames = {

null, "STAR", "LETTER", "DIGIT", "SPLITTER", "WS"

};

public static readonly IVocabulary DefaultVocabulary = new Vocabulary(\_LiteralNames, \_SymbolicNames);

[NotNull]

public override IVocabulary Vocabulary

{

get

{

return DefaultVocabulary;

}

}

public override string GrammarFileName { get { return "CommentsFortran.g4"; } }

public override string[] RuleNames { get { return ruleNames; } }

public override string[] ChannelNames { get { return channelNames; } }

public override string[] ModeNames { get { return modeNames; } }

public override string SerializedAtn { get { return new string(\_serializedATN); } }

static CommentsFortranLexer() {

decisionToDFA = new DFA[\_ATN.NumberOfDecisions];

for (int i = 0; i < \_ATN.NumberOfDecisions; i++) {

decisionToDFA[i] = new DFA(\_ATN.GetDecisionState(i), i);

}

}

private static char[] \_serializedATN = {

'\x3', '\x608B', '\xA72A', '\x8133', '\xB9ED', '\x417C', '\x3BE7', '\x7786',

'\x5964', '\x2', '\a', '\x1A', '\b', '\x1', '\x4', '\x2', '\t', '\x2',

'\x4', '\x3', '\t', '\x3', '\x4', '\x4', '\t', '\x4', '\x4', '\x5', '\t',

'\x5', '\x4', '\x6', '\t', '\x6', '\x3', '\x2', '\x3', '\x2', '\x3', '\x3',

'\x3', '\x3', '\x3', '\x4', '\x3', '\x4', '\x3', '\x5', '\x3', '\x5',

'\x3', '\x6', '\x5', '\x6', '\x17', '\n', '\x6', '\x3', '\x6', '\x3',

'\x6', '\x2', '\x2', '\a', '\x3', '\x3', '\x5', '\x4', '\a', '\x5', '\t',

'\x6', '\v', '\a', '\x3', '\x2', '\x4', '\x4', '\x2', '\x43', '\\', '\x63',

'|', '\t', '\x2', '\"', '#', '\*', '+', '-', '\x31', '>', '\x41', ']',

'\_', '}', '}', '\x7F', '\x7F', '\x2', '\x1A', '\x2', '\x3', '\x3', '\x2',

'\x2', '\x2', '\x2', '\x5', '\x3', '\x2', '\x2', '\x2', '\x2', '\a', '\x3',

'\x2', '\x2', '\x2', '\x2', '\t', '\x3', '\x2', '\x2', '\x2', '\x2', '\v',

'\x3', '\x2', '\x2', '\x2', '\x3', '\r', '\x3', '\x2', '\x2', '\x2', '\x5',

'\xF', '\x3', '\x2', '\x2', '\x2', '\a', '\x11', '\x3', '\x2', '\x2',

'\x2', '\t', '\x13', '\x3', '\x2', '\x2', '\x2', '\v', '\x16', '\x3',

'\x2', '\x2', '\x2', '\r', '\xE', '\a', ',', '\x2', '\x2', '\xE', '\x4',

'\x3', '\x2', '\x2', '\x2', '\xF', '\x10', '\t', '\x2', '\x2', '\x2',

'\x10', '\x6', '\x3', '\x2', '\x2', '\x2', '\x11', '\x12', '\x4', '\x32',

';', '\x2', '\x12', '\b', '\x3', '\x2', '\x2', '\x2', '\x13', '\x14',

'\t', '\x3', '\x2', '\x2', '\x14', '\n', '\x3', '\x2', '\x2', '\x2', '\x15',

'\x17', '\a', '\xF', '\x2', '\x2', '\x16', '\x15', '\x3', '\x2', '\x2',

'\x2', '\x16', '\x17', '\x3', '\x2', '\x2', '\x2', '\x17', '\x18', '\x3',

'\x2', '\x2', '\x2', '\x18', '\x19', '\a', '\f', '\x2', '\x2', '\x19',

'\f', '\x3', '\x2', '\x2', '\x2', '\x4', '\x2', '\x16', '\x2',

};

public static readonly ATN \_ATN =

new ATNDeserializer().Deserialize(\_serializedATN);

}

CommentsFortranParser.cs

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated by a tool.

// ANTLR Version: 4.9.2

//

// Changes to this file may cause incorrect behavior and will be lost if

// the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

// Generated from CommentsFortran.g4 by ANTLR 4.9.2

// Unreachable code detected

#pragma warning disable 0162

// The variable '...' is assigned but its value is never used

#pragma warning disable 0219

// Missing XML comment for publicly visible type or member '...'

#pragma warning disable 1591

// Ambiguous reference in cref attribute

#pragma warning disable 419

using System;

using System.Collections;

using System;

using System.IO;

using System.Text;

using System.Diagnostics;

using System.Collections.Generic;

using Antlr4.Runtime;

using Antlr4.Runtime.Atn;

using Antlr4.Runtime.Misc;

using Antlr4.Runtime.Tree;

using DFA = Antlr4.Runtime.Dfa.DFA;

[System.CodeDom.Compiler.GeneratedCode("ANTLR", "4.9.2")]

[System.CLSCompliant(false)]

public partial class CommentsFortranParser : Parser {

protected static DFA[] decisionToDFA;

protected static PredictionContextCache sharedContextCache = new PredictionContextCache();

public const int

STAR=1, LETTER=2, DIGIT=3, SPLITTER=4, WS=5;

public const int

RULE\_prog = 0, RULE\_comment = 1;

public static readonly string[] ruleNames = {

"prog", "comment"

};

private static readonly string[] \_LiteralNames = {

null, "'\*'"

};

private static readonly string[] \_SymbolicNames = {

null, "STAR", "LETTER", "DIGIT", "SPLITTER", "WS"

};

public static readonly IVocabulary DefaultVocabulary = new Vocabulary(\_LiteralNames, \_SymbolicNames);

[NotNull]

public override IVocabulary Vocabulary

{

get

{

return DefaultVocabulary;

}

}

public override string GrammarFileName { get { return "CommentsFortran.g4"; } }

public override string[] RuleNames { get { return ruleNames; } }

public override string SerializedAtn { get { return new string(\_serializedATN); } }

static CommentsFortranParser() {

decisionToDFA = new DFA[\_ATN.NumberOfDecisions];

for (int i = 0; i < \_ATN.NumberOfDecisions; i++) {

decisionToDFA[i] = new DFA(\_ATN.GetDecisionState(i), i);

}

}

public CommentsFortranParser(ITokenStream input) : this(input, Console.Out, Console.Error) { }

public CommentsFortranParser(ITokenStream input, TextWriter output, TextWriter errorOutput)

: base(input, output, errorOutput)

{

Interpreter = new ParserATNSimulator(this, \_ATN, decisionToDFA, sharedContextCache);

}

public partial class ProgContext : ParserRuleContext {

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode Eof() { return GetToken(CommentsFortranParser.Eof, 0); }

[System.Diagnostics.DebuggerNonUserCode] public CommentContext[] comment() {

return GetRuleContexts<CommentContext>();

}

[System.Diagnostics.DebuggerNonUserCode] public CommentContext comment(int i) {

return GetRuleContext<CommentContext>(i);

}

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode[] WS() { return GetTokens(CommentsFortranParser.WS); }

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode WS(int i) {

return GetToken(CommentsFortranParser.WS, i);

}

public ProgContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

public override int RuleIndex { get { return RULE\_prog; } }

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

ICommentsFortranListener typedListener = listener as ICommentsFortranListener;

if (typedListener != null) typedListener.EnterProg(this);

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

ICommentsFortranListener typedListener = listener as ICommentsFortranListener;

if (typedListener != null) typedListener.ExitProg(this);

}

}

[RuleVersion(0)]

public ProgContext prog() {

ProgContext \_localctx = new ProgContext(Context, State);

EnterRule(\_localctx, 0, RULE\_prog);

int \_la;

try {

int \_alt;

EnterOuterAlt(\_localctx, 1);

{

State = 9;

ErrorHandler.Sync(this);

\_alt = Interpreter.AdaptivePredict(TokenStream,0,Context);

while ( \_alt!=2 && \_alt!=global::Antlr4.Runtime.Atn.ATN.INVALID\_ALT\_NUMBER ) {

if ( \_alt==1 ) {

{

{

State = 4;

comment();

State = 5;

Match(WS);

}

}

}

State = 11;

ErrorHandler.Sync(this);

\_alt = Interpreter.AdaptivePredict(TokenStream,0,Context);

}

State = 13;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

if (\_la==STAR) {

{

State = 12;

comment();

}

}

State = 15;

Match(Eof);

}

}

catch (RecognitionException re) {

\_localctx.exception = re;

ErrorHandler.ReportError(this, re);

ErrorHandler.Recover(this, re);

}

finally {

ExitRule();

}

return \_localctx;

}

public partial class CommentContext : ParserRuleContext {

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode[] STAR() { return GetTokens(CommentsFortranParser.STAR); }

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode STAR(int i) {

return GetToken(CommentsFortranParser.STAR, i);

}

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode[] LETTER() { return GetTokens(CommentsFortranParser.LETTER); }

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode LETTER(int i) {

return GetToken(CommentsFortranParser.LETTER, i);

}

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode[] DIGIT() { return GetTokens(CommentsFortranParser.DIGIT); }

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode DIGIT(int i) {

return GetToken(CommentsFortranParser.DIGIT, i);

}

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode[] SPLITTER() { return GetTokens(CommentsFortranParser.SPLITTER); }

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode SPLITTER(int i) {

return GetToken(CommentsFortranParser.SPLITTER, i);

}

public CommentContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

public override int RuleIndex { get { return RULE\_comment; } }

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

ICommentsFortranListener typedListener = listener as ICommentsFortranListener;

if (typedListener != null) typedListener.EnterComment(this);

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

ICommentsFortranListener typedListener = listener as ICommentsFortranListener;

if (typedListener != null) typedListener.ExitComment(this);

}

}

[RuleVersion(0)]

public CommentContext comment() {

CommentContext \_localctx = new CommentContext(Context, State);

EnterRule(\_localctx, 2, RULE\_comment);

int \_la;

try {

EnterOuterAlt(\_localctx, 1);

{

State = 17;

Match(STAR);

State = 21;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

while ((((\_la) & ~0x3f) == 0 && ((1L << \_la) & ((1L << STAR) | (1L << LETTER) | (1L << DIGIT) | (1L << SPLITTER))) != 0)) {

{

{

State = 18;

\_la = TokenStream.LA(1);

if ( !((((\_la) & ~0x3f) == 0 && ((1L << \_la) & ((1L << STAR) | (1L << LETTER) | (1L << DIGIT) | (1L << SPLITTER))) != 0)) ) {

ErrorHandler.RecoverInline(this);

}

else {

ErrorHandler.ReportMatch(this);

Consume();

}

}

}

State = 23;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

}

}

}

catch (RecognitionException re) {

\_localctx.exception = re;

ErrorHandler.ReportError(this, re);

ErrorHandler.Recover(this, re);

}

finally {

ExitRule();

}

return \_localctx;

}

private static char[] \_serializedATN = {

'\x3', '\x608B', '\xA72A', '\x8133', '\xB9ED', '\x417C', '\x3BE7', '\x7786',

'\x5964', '\x3', '\a', '\x1B', '\x4', '\x2', '\t', '\x2', '\x4', '\x3',

'\t', '\x3', '\x3', '\x2', '\x3', '\x2', '\x3', '\x2', '\a', '\x2', '\n',

'\n', '\x2', '\f', '\x2', '\xE', '\x2', '\r', '\v', '\x2', '\x3', '\x2',

'\x5', '\x2', '\x10', '\n', '\x2', '\x3', '\x2', '\x3', '\x2', '\x3',

'\x3', '\x3', '\x3', '\a', '\x3', '\x16', '\n', '\x3', '\f', '\x3', '\xE',

'\x3', '\x19', '\v', '\x3', '\x3', '\x3', '\x2', '\x2', '\x4', '\x2',

'\x4', '\x2', '\x3', '\x3', '\x2', '\x3', '\x6', '\x2', '\x1B', '\x2',

'\v', '\x3', '\x2', '\x2', '\x2', '\x4', '\x13', '\x3', '\x2', '\x2',

'\x2', '\x6', '\a', '\x5', '\x4', '\x3', '\x2', '\a', '\b', '\a', '\a',

'\x2', '\x2', '\b', '\n', '\x3', '\x2', '\x2', '\x2', '\t', '\x6', '\x3',

'\x2', '\x2', '\x2', '\n', '\r', '\x3', '\x2', '\x2', '\x2', '\v', '\t',

'\x3', '\x2', '\x2', '\x2', '\v', '\f', '\x3', '\x2', '\x2', '\x2', '\f',

'\xF', '\x3', '\x2', '\x2', '\x2', '\r', '\v', '\x3', '\x2', '\x2', '\x2',

'\xE', '\x10', '\x5', '\x4', '\x3', '\x2', '\xF', '\xE', '\x3', '\x2',

'\x2', '\x2', '\xF', '\x10', '\x3', '\x2', '\x2', '\x2', '\x10', '\x11',

'\x3', '\x2', '\x2', '\x2', '\x11', '\x12', '\a', '\x2', '\x2', '\x3',

'\x12', '\x3', '\x3', '\x2', '\x2', '\x2', '\x13', '\x17', '\a', '\x3',

'\x2', '\x2', '\x14', '\x16', '\t', '\x2', '\x2', '\x2', '\x15', '\x14',

'\x3', '\x2', '\x2', '\x2', '\x16', '\x19', '\x3', '\x2', '\x2', '\x2',

'\x17', '\x15', '\x3', '\x2', '\x2', '\x2', '\x17', '\x18', '\x3', '\x2',

'\x2', '\x2', '\x18', '\x5', '\x3', '\x2', '\x2', '\x2', '\x19', '\x17',

'\x3', '\x2', '\x2', '\x2', '\x5', '\v', '\xF', '\x17',

};

public static readonly ATN \_ATN =

new ATNDeserializer().Deserialize(\_serializedATN);

}

CommentAnalyzerErrorListener.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Antlr4.Runtime;

namespace comment\_analyzer

{

class CommentAnalyzerErrorListener : BaseErrorListener

{

public List<string> errorsList = new List<string>();

public override void SyntaxError(TextWriter output, IRecognizer recognizer, IToken offendingSymbol, int line, int charPositionInLine, string msg, RecognitionException e)

{

errorsList.Add("Error(" + line + ":" + charPositionInLine + "): " + msg);

//base.SyntaxError(output, recognizer, offendingSymbol, line, charPositionInLine, msg, e);

}

}

}