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 2](#_Toc88689470)

[1. Постановка задачи 3](#_Toc88689471)

[2. Разработка грамматики 4](#_Toc88689472)

[3. Грамматика для ANTLR 6](#_Toc88689473)

[4. Внедрение грамматики в интерфейс, созданной в ANTLR 7](#_Toc88689474)

[5. Внедрение грамматики в интерфейс, созданный Flex & Bison 9](#_Toc88689475)

[6. Тестовые примеры 11](#_Toc88689476)

[7. Листинг программы 15](#_Toc88689477)

[Заключение 16](#_Toc88689478)

[Список использованных источников 18](#_Toc88689479)

[Приложение А 19](#_Toc88689480)

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. Внедрение грамматики в интерфейс, созданный Flex & Bison

Flex и Bison – две программы, которые в паре способны создать синтаксический анализатор. Flex используют для создания токенов, которые затем в Bison используются для реализации грамматики. Кроме создания токенов и грамматики необходимо также создать обертку для анализатора путем добавления кода на языке C/C++ в файле с грамматикой. Собирая по отдельности файлы с токенами для Flex и файлы с грамматикой для Bison получаются файлы исходного кода, написанные на языке C. Компилируя полученные файлы любым компилятором для С/С++ на выходе получается исполняемый файл, который уже в состоянии провести синтаксический анализ на основе разработанной грамматики. Исходный код для разработанного мною анализатора предоставлен в приложении А.

Сборка файлов анализатора производилась в ОС Windows 10, поэтому для преобразования файлов Flex и Bison необходимо скачать программы win\_flex.exe и win\_bison.exe, которые доступны по адресу <https://sourceforge.net/projects/winflexbison/>. После загрузки программы необходимо разместить в той же папке, что и файлы разработанной грамматики.

Сборка файла токенов CommentAnalyzer.l выполняется с помощью команды в консоли:

win\_flex.exe CommentAnalyzer.l

Сборка файла грамматики CommentAnalyzer.y выполняется с помощью команды в консоли:

win\_bison.exe –d CommentAnalyzer.y

В итоге получаем файл токенов lex.yy.c и два файла, содержащих исходный код анализатора CommentAnalyzer.tab.c и CommentAnalyzer.tab.h. Теперь требуется с помощью следующей консольной команды осуществить компиляцию полученных файлов с исполняемый файл:

g++ CommentAnalyzer.tab.c lex.yy.c –o CommentAnalyzer.exe –lws2\_32

Так как была необходимость внедрения грамматики в уже существующий интерфейс, реализованный на языке C#, мною было принято решение использовать получение анализатором текста с помощью сокетов. То есть анализатор представляет собой внешнюю программу, которая может принимать сообщения от других программ и возвращать им результат.

Интерфейс также требовал доработки для того, чтобы уметь передавать и получать данные с помощью сокетов. Мною был реализован метод CommandCheckFlexBison, который как раз и выполняет данную процедуру. Весь исходный код интерфейса с доработками представлен в приложении А.

1. Тестовые примеры

На приведенных ниже рисунках 1 - 8 показаны примеры работы разработанной программы:

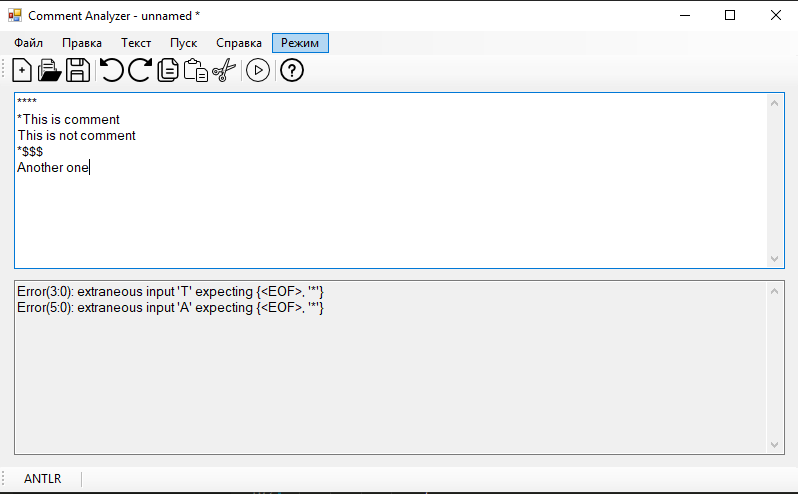


Рисунок 1 – Тестовый пример 1

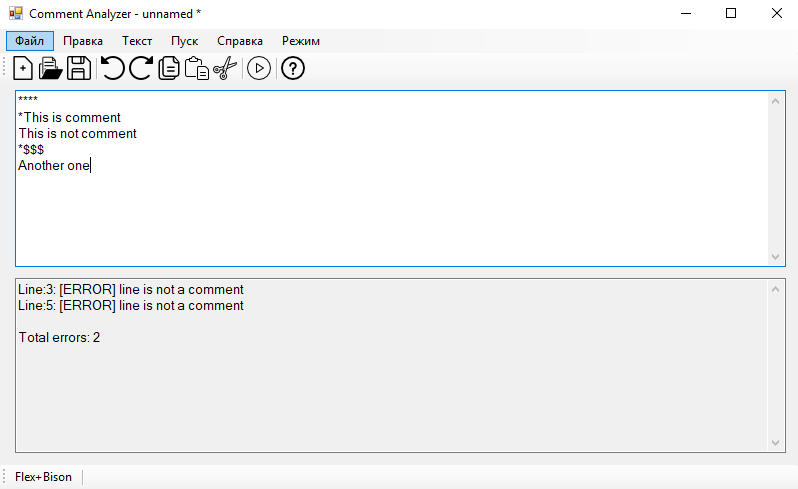


Рисунок 2 – Тестовый пример 2

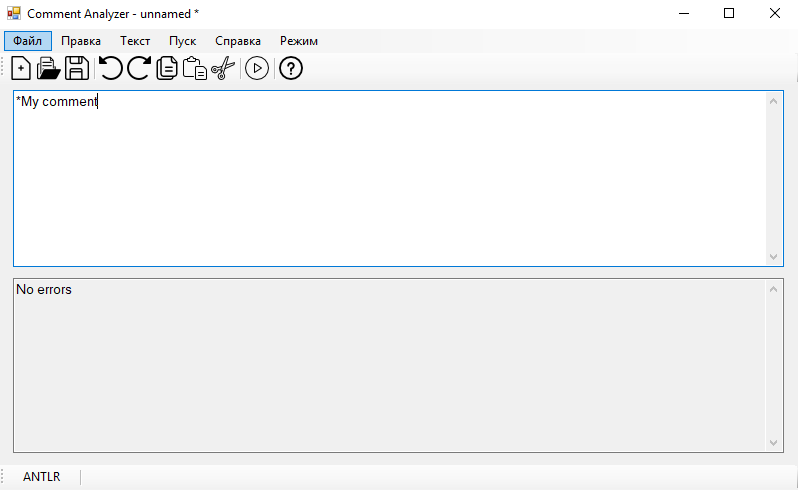


Рисунок 3 – Тестовый пример 3

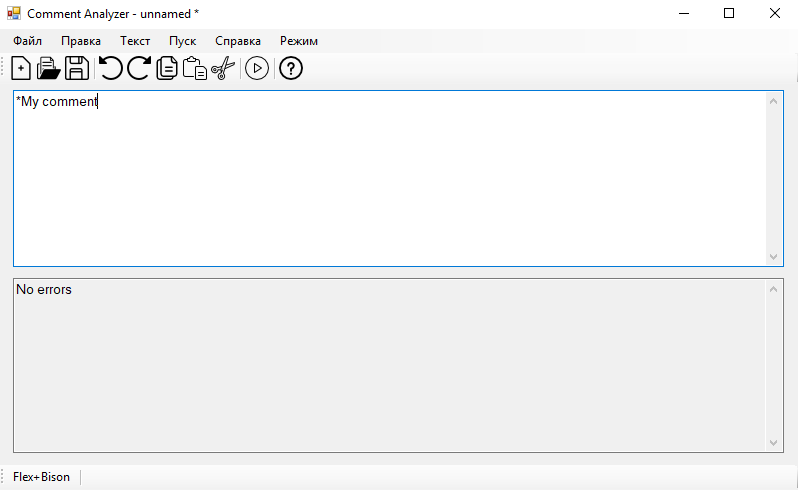


Рисунок 4 – Тестовый пример 4

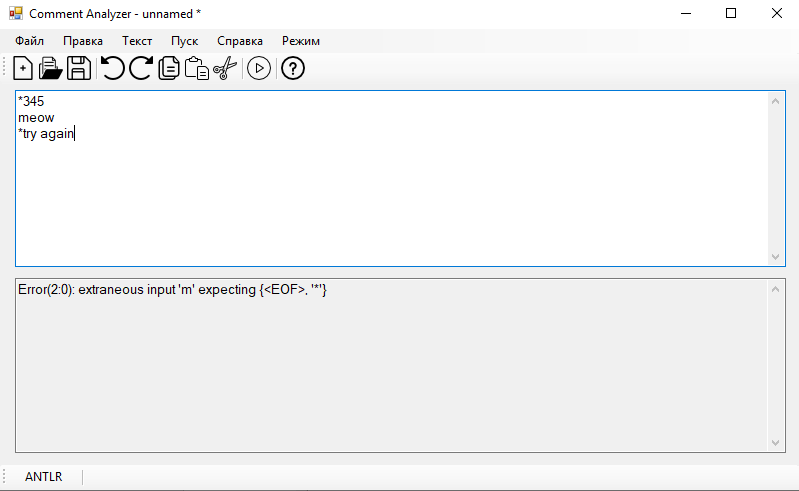


Рисунок 5 – Тестовый пример 5

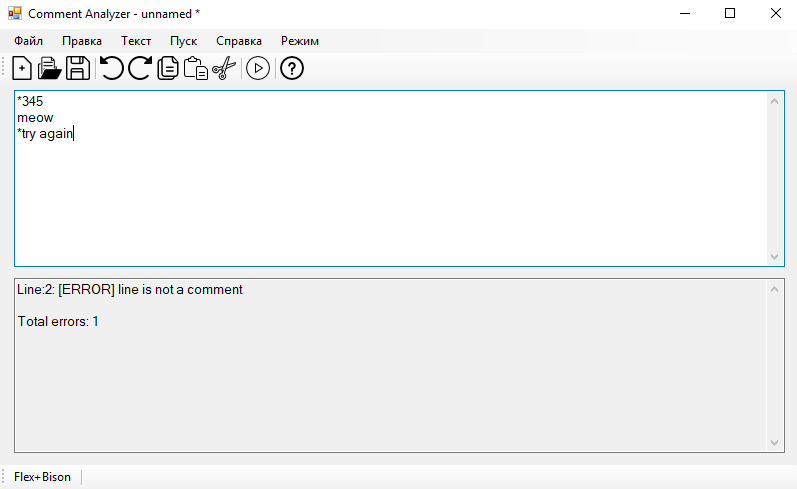


Рисунок 6 – Тестовый пример 6

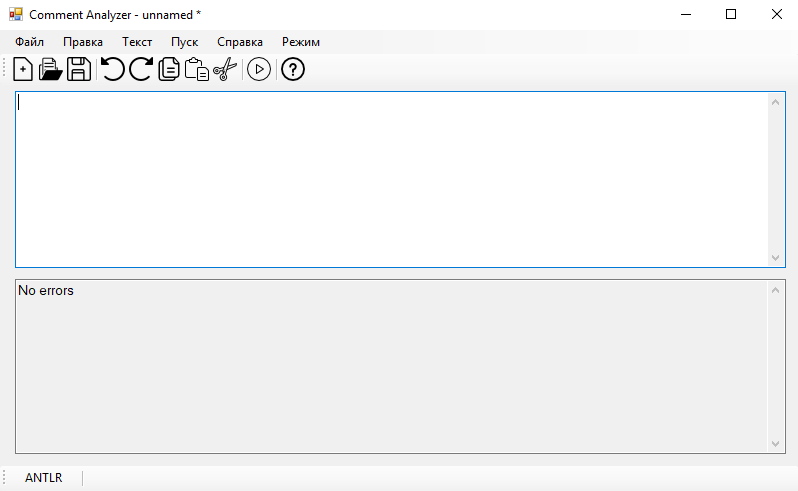


Рисунок 7 – Тестовый пример 7

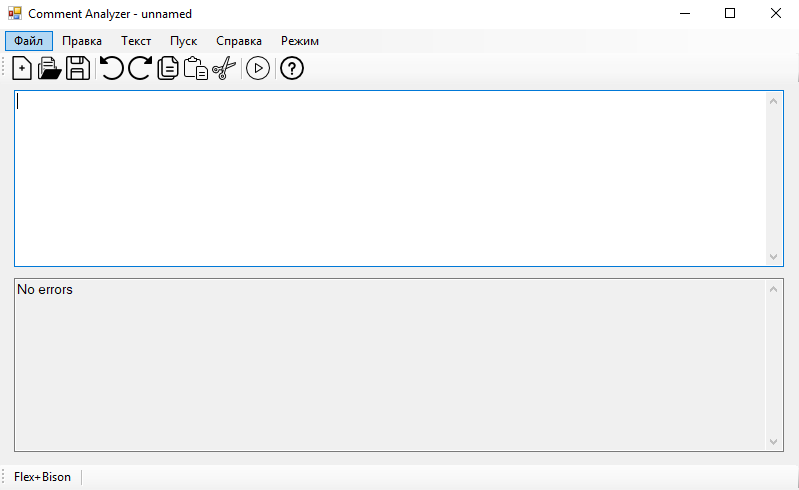


Рисунок 8 – Тестовый пример 8

1. Листинг программы

Листинг программной части сгенерированного с помощью ANTLR анализатора, исходный код для Flex и Bison, реализация обработчика ошибок для ANTLR и дополненный исходный код интерфейса приведены в Приложении А.

Заключение

В результате выполнения расчетно-графической работы была реализована грамматика комментариев языка Fortran для ANTLR и для Flex&Bison. Далее был получен модуль синтаксического анализатора на языке C# путем компиляции грамматики в программе ANTLR. Кроме этого был реализован синтаксический анализатор с помощью Flex и Bison на языке C. Он представляет собой внешнюю программу, способную обмениваться информацией с другими программами через сокеты. Затем к ранее разработанному графическому интерфейсу на C# был подключен модуль анализатора с помощью библиотеки Antlr4.Runtime. Также интерфейс был дополнен кодом, позволяющим через сокеты обмениваться данными с анализатором на основе Flex и Bison. Работа анализатора была протестирована на разных данных.

В качестве выводов по опыту работы с ANTLR, FLEX и BISON могу сказать следующее: данные программы крайне удобны при проектировании синтаксических анализаторов, так как позволяют избежать ошибок реализации анализатора на целевом языке программирования. При их использовании разработчик может уделить больше времени 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 данных программ хочется привести ситуацию, при которой анализатор не в состоянии исправить незначительные ошибки или опечатки. Такая ситуация наблюдалась в ANTLR. 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. Кроме того, невозможность реализации анализатора на языке C# в случае с Flex и Bison вынудила прийти к существенному усложнению разработки обертки для анализатора.

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).
5. Flex & Bison / John R. Levine: O’Reilly Media, 2009. 294 c.
6. Парсим Python код с помощью Flex и Bison // Хабр URL: https://habr.com/ru/post/141756/ (дата обращения: 14.11.2021).

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);

}

AntlrErrorListener.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 AntlrErrorListener : 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);

}

}

}

CommentAnalyzer.y

%{

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Need to link with Ws2\_32.lib

#pragma comment (lib, "Ws2\_32.lib")

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "777"

#define ERROR\_MAX\_LENGTH 160

*int* lineCounter = 1;

*int* errorType = 0;

extern *int* yylex();

extern *int* yyparse();

extern FILE\* yyin;

*void* yyerror(const *char*\* *s*);

%}

%union {

    int ival;

    float fval;

}

%token T\_STAR

%token T\_DIGIT

%token T\_LETTER

%token T\_SPLITTER

%token T\_NEWLINE

%token T\_QUIT

%start prog

%%

prog:

    |comment prog

;

comment: T\_NEWLINE  {lineCounter++;}

    |   T\_QUIT T\_NEWLINE    {exit(0);}

    |   T\_STAR correct T\_NEWLINE    {lineCounter++;}

    |   T\_LETTER notcomment error T\_NEWLINE {lineCounter++;}

    |   T\_DIGIT notcomment error T\_NEWLINE  {lineCounter++;}

    |   T\_SPLITTER notcomment error T\_NEWLINE   {lineCounter++;}

;

correct:

    |   T\_LETTER correct

    |   T\_DIGIT correct

    |   T\_SPLITTER correct

    |   T\_STAR correct

;

notcomment: {errorType = 1;}

%%

*int* errorsCount = 0;

*char*\*\* errors;

*int* \_\_cdecl main(*void*) {

    WSADATA wsaData;

    int iResult;

    SOCKET ListenSocket = INVALID\_SOCKET;

    SOCKET ClientSocket = INVALID\_SOCKET;

    struct addrinfo \*result = NULL;

    struct addrinfo hints;

    int iSendResult;

    char recvbuf[DEFAULT\_BUFLEN];

    int recvbuflen = DEFAULT\_BUFLEN;

    // Initialize Winsock

    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);

    if (iResult != 0) {

        printf("WSAStartup failed with error: %d\n", iResult);

        return 1;

    }

    ZeroMemory(&hints, sizeof(hints));

    hints.ai\_family = AF\_INET;

    hints.ai\_socktype = SOCK\_STREAM;

    hints.ai\_protocol = IPPROTO\_TCP;

    hints.ai\_flags = AI\_PASSIVE;

    // Resolve the server address and port

    iResult = getaddrinfo(NULL, DEFAULT\_PORT, &hints, &result);

    if ( iResult != 0 ) {

        printf("getaddrinfo failed with error: %d\n", iResult);

        WSACleanup();

        return 1;

    }

    // Create a SOCKET for connecting to server

    ListenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

    if (ListenSocket == INVALID\_SOCKET) {

        printf("socket failed with error: %ld\n", WSAGetLastError());

        freeaddrinfo(result);

        WSACleanup();

        return 1;

    }

        // Setup the TCP listening socket

    iResult = bind( ListenSocket, result->ai\_addr, (int)result->ai\_addrlen);

    if (iResult == SOCKET\_ERROR) {

        printf("bind failed with error: %d\n", WSAGetLastError());

        freeaddrinfo(result);

        closesocket(ListenSocket);

        WSACleanup();

        return 1;

    }

        freeaddrinfo(result);

    iResult = listen(ListenSocket, SOMAXCONN);

    if (iResult == SOCKET\_ERROR) {

        printf("listen failed with error: %d\n", WSAGetLastError());

        closesocket(ListenSocket);

        WSACleanup();

        return 1;

    }

    printf("Server opened at 127.0.0.1:%s\n", DEFAULT\_PORT);

    do {

        // Accept a client socket

        ClientSocket = accept(ListenSocket, NULL, NULL);

        if (ClientSocket == INVALID\_SOCKET) {

            printf("accept failed with error: %d\n", WSAGetLastError());

            closesocket(ListenSocket);

            WSACleanup();

            return 1;

        }

        // No longer need server socket

        //closesocket(ListenSocket);

        // Receive until the peer shuts down the connection

        do {

            iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);

            if (iResult > 0) {

                printf("Bytes received: %d\n", iResult);

                printf("Message: %s\n", recvbuf);

                FILE\* input = fopen("input.txt", "w");

                fprintf(input, "%s", recvbuf);

                fclose(input);

                yyin = fopen("input.txt", "r");

                do {

                    yyparse();

                }while(!feof(yyin));

                char error[ERROR\_MAX\_LENGTH \* errorsCount \* sizeof(char)];

                strcpy(error, (char\*)"");

                if (errorsCount - 1 > 0) {

                    for(int i = 0; i < errorsCount; i++) {

                        if (i == 0)

                            strcpy(error, errors[i]);

                        else

                            strcat(error, errors[i]);

                    }

                    strcat(error, "\r\n");

                    strcat(error, "Total errors: ");

                    char\* totalErrors = (char\*) calloc (sizeof(char), 10);

                    itoa(errorsCount - 1, totalErrors, 10);

                    strcat(error, totalErrors);

                } else {

                    strcpy(error, (char\*)"No errors\r\n");

                }

                // Echo the buffer back to the sender

                iSendResult = send( ClientSocket, error, ERROR\_MAX\_LENGTH \* errorsCount \* sizeof(char), 0 );

                if (iSendResult == SOCKET\_ERROR) {

                    printf("send failed with error: %d\n", WSAGetLastError());

                    closesocket(ClientSocket);

                    WSACleanup();

                    return 1;

                }

                printf("Bytes sent: %d\n", iSendResult);

            }

            else if (iResult == 0)

                printf("Waiting for new data...\n");

            else  {

                printf("recv failed with error: %d\n", WSAGetLastError());

                closesocket(ClientSocket);

                WSACleanup();

                return 1;

            }

            errors = NULL;

            errorsCount = 0;

            lineCounter = 1;

            errorType = 0;

            for (int i = 0; i < recvbuflen; i++){

                recvbuf[i] = (char)0;

            }

        } while (iResult > 0);

        // shutdown the connection since we're done

        iResult = shutdown(ClientSocket, SD\_SEND);

        if (iResult == SOCKET\_ERROR) {

            printf("shutdown failed with error: %d\n", WSAGetLastError());

            closesocket(ClientSocket);

            WSACleanup();

            return 1;

        }

    }while(true);

    // cleanup

    closesocket(ClientSocket);

    WSACleanup();

    exit(0);

}

void yyerror(const char\* s) {

    if (errors == NULL) {

        errors = (char\*\*) calloc (sizeof(char\*), ++errorsCount);

    } else {

        errors = (char\*\*) realloc (errors, sizeof(char\*)\*(++errorsCount));

    }

    errors[errorsCount - 1] = (char\*) calloc (sizeof(char), 200);

    if (errorType == 1) {

        strcpy(errors[errorsCount - 1], "");

        char\* line = (char\*) calloc (sizeof(char), 10);

        itoa(lineCounter, line, 10);

        strcat(errors[errorsCount - 1], "Line:");

        strcat(errors[errorsCount - 1], (const char\*)line);

        strcat(errors[errorsCount - 1], ": [ERROR] line is not a comment\r\n");

    }

    errorType = 0;

    //exit(1);

}

CommentAnalyzer.l

%option noyywrap

%{

#include <stdio.h>

#define YY\_DECL int yylex()

#include "CommentAnalyzer.tab.h"

%}

%%

["\t"]  ; // ignore all whitespace

"\n"    {return T\_NEWLINE;}

"exit"    {return T\_QUIT;}

"quit"    {return T\_QUIT;}

"\*"                     {return T\_STAR;}

[0-9]                   {return T\_DIGIT;}

[a-zA-Z]                {return T\_LETTER;}

[.,!?(){}+-=<> ]        {return T\_SPLITTER;}

%%

Commands.cs

using Antlr4.Runtime;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Net;

using System.Net.Sockets;

namespace comment\_analyzer

{

public class Commands

{

public void CommandCreate()

{

if (StaticData.unsaved)

{

StaticData.currentData = StaticData.mainForm.TextBox.Text;

var saveBeforeCloseWindow = new SaveBeforeCloseForm();

saveBeforeCloseWindow.ShowDialog();

}

StaticData.dialogService.FilePath = "";

StaticData.currentData = "";

StaticData.mainForm.TextBox.Text = StaticData.currentData;

StaticData.mainForm.Heading = "Language Processor - unnamed";

}

public void CommandOpen()

{

if (StaticData.unsaved)

{

StaticData.currentData = StaticData.mainForm.TextBox.Text;

var saveBeforeCloseWindow = new SaveBeforeCloseForm();

saveBeforeCloseWindow.ShowDialog();

}

StaticData.dialogService.OpenFileDialog();

StaticData.currentData = StaticData.fileService.ReadFile(StaticData.dialogService.FilePath);

StaticData.mainForm.TextBox.Text = StaticData.currentData;

StaticData.mainForm.Heading = "Language Processor";

if (StaticData.dialogService.FilePath != null || StaticData.dialogService.FilePath != "")

StaticData.mainForm.Heading += " - " + StaticData.dialogService.FilePath;

else

StaticData.mainForm.Heading += " - unnamed";

StaticData.unsaved = false;

}

public void CommandSave()

{

StaticData.currentData = StaticData.mainForm.TextBox.Text;

if (StaticData.dialogService.FilePath == null)

{

StaticData.dialogService.SaveFileDialog();

StaticData.fileService.SaveFile(StaticData.dialogService.FilePath, StaticData.currentData);

}

else

{

StaticData.fileService.SaveFile(StaticData.dialogService.FilePath, StaticData.currentData);

}

StaticData.unsaved = false;

StaticData.mainForm.Heading = "Language Processor - " + StaticData.dialogService.FilePath;

}

public void CommandSaveAs()

{

StaticData.currentData = StaticData.mainForm.TextBox.Text;

StaticData.dialogService.SaveFileDialog();

StaticData.fileService.SaveFile(StaticData.dialogService.FilePath, StaticData.currentData);

StaticData.mainForm.Heading = "Language Processor - " + StaticData.dialogService.FilePath;

StaticData.unsaved = false;

}

public void CommandUndo()

{

if (StaticData.undoStack.Count > 0)

{

StaticData.redoStack.Push(StaticData.mainForm.TextBox.Text);

string newValue = StaticData.undoStack.Pop();

StaticData.mainForm.TextBox.Text = newValue;

}

}

public void CommandRedo()

{

if (StaticData.redoStack.Count > 0)

{

StaticData.undoStack.Push(StaticData.mainForm.TextBox.Text);

string newValue = StaticData.redoStack.Pop();

StaticData.mainForm.TextBox.Text = newValue;

}

}

public void CommandCopy()

{

if (StaticData.mainForm.TextBox.SelectionLength > 0)

StaticData.mainForm.TextBox.Copy();

}

public void CommandPaste()

{

if (Clipboard.GetDataObject().GetDataPresent(DataFormats.Text) == true)

{

if (StaticData.mainForm.TextBox.SelectionLength > 0)

{

StaticData.mainForm.TextBox.SelectionStart = StaticData.mainForm.TextBox.SelectionStart + StaticData.mainForm.TextBox.SelectionLength;

}

StaticData.mainForm.TextBox.Paste();

}

}

public void CommandCut()

{

if (StaticData.mainForm.TextBox.SelectedText != "")

StaticData.mainForm.TextBox.Cut();

}

public void CommandDelete()

{

int StartPosDel = StaticData.mainForm.TextBox.SelectionStart;

int LenSelection = StaticData.mainForm.TextBox.SelectionLength;

StaticData.mainForm.TextBox.Text = StaticData.mainForm.TextBox.Text.Remove(StartPosDel, LenSelection);

}

public void CommandSelectAll()

{

StaticData.mainForm.TextBox.SelectAll();

}

public void CommandHelp()

{

Help.ShowHelp(null, StaticData.prePath + "help/help1.html");

}

public void CommandCheck()

{

switch (StaticData.mode) {

case "Automata":

CommandCheckAutomata();

break;

case "ANTLR":

CommandCheckANTLR();

break;

case "Flex+Bison":

CommandCheckFlexBison();

break;

default:

break;

}

}

public void CommandCheckAutomata()

{

string[] errors = { "Ошибок не обнаружено",

"Пустая строка",

"Строка не является комментарием(начинается не с символа \*)"};

string warning = "Обнаружена последовательность символов не из алфавита '";

int line = 0;

int status = 0;

bool hasErrors = false;

int errorsCount = 0;

int warnCount = 0;

string[] strings = StaticData.mainForm.TextBox.Text.Split('\n');

for (int i = 0; i < strings.Length; i++)

{

strings[i] = strings[i].TrimEnd('\r');

}

List<string> wrongSymbols = new List<string>();

List<int> wrongPositions = new List<int>();

Automata automata = new Automata();

StaticData.mainForm.ResultsTextBox.Text = "";

for (line = 0; line < strings.Length; line++)

{

status = automata.analyzeLine(strings[line], ref wrongSymbols, ref wrongPositions);

if (status >= 3)

{

for (int j = 0; wrongSymbols.Count > j; j++)

{

warnCount++;

StaticData.mainForm.ResultsTextBox.Text += "Строка " + (line + 1).ToString() + ": [Предуп] " +

warning + wrongSymbols[j] + "'(начало с " + (wrongPositions[j] + 1) + " символа)" + Environment.NewLine;

}

status -= 3;

wrongSymbols.Clear();

wrongPositions.Clear();

}

if (status != 0)

{

StaticData.mainForm.ResultsTextBox.Text += "Строка " + (line + 1).ToString() + ": [Ошибка] " +

errors[status] + Environment.NewLine;

hasErrors = true;

errorsCount++;

}

}

StaticData.mainForm.ResultsTextBox.Text += Environment.NewLine;

if (warnCount > 0)

{

StaticData.mainForm.ResultsTextBox.Text += "Предупреждений: " + warnCount + Environment.NewLine;

}

if (!hasErrors)

{

StaticData.mainForm.ResultsTextBox.Text += "Ошибок не обнаружено" + Environment.NewLine;

}

else

{

StaticData.mainForm.ResultsTextBox.Text += "Ошибок: " + errorsCount + Environment.NewLine;

}

}

public void CommandCheckANTLR()

{

// 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

AntlrErrorListener customListener = new AntlrErrorListener();

// 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";

}

}

public void CommandCheckFlexBison ()

{

String source = StaticData.mainForm.TextBox.Text;

if (source.Length == 0)

{

source = "\*\*\*";

}

int port = 777;

string address = "127.0.0.1";

try

{

IPEndPoint ipPoint = new IPEndPoint(IPAddress.Parse(address), port);

Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

// подключаемся к удаленному хосту

socket.Connect(ipPoint);

byte[] data = Encoding.ASCII.GetBytes(source);

socket.Send(data);

// получаем ответ

data = new byte[256]; // буфер для ответа

StringBuilder builder = new StringBuilder();

int bytes = 0; // количество полученных байт

do

{

bytes = socket.Receive(data, data.Length, 0);

builder.Append(Encoding.ASCII.GetString(data, 0, bytes));

}

while (socket.Available > 0);

//Console.WriteLine("ответ сервера: " + builder.ToString());

StaticData.mainForm.ResultsTextBox.Text = builder.ToString();

// закрываем сокет

socket.Shutdown(SocketShutdown.Both);

socket.Close();

}

catch (Exception ex)

{

//Console.WriteLine(ex.Message);

StaticData.mainForm.ResultsTextBox.Text = ex.Message.ToString();

}

}

}

}

CommentAnalyzerForm.cs

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;

using Antlr4.Runtime;

namespace comment\_analyzer

{

public partial class CommentAnalyzerForm : Form

{

//wrappers for elements of form

public System.Windows.Forms.TextBox TextBox

{

get { return this.textBox1; }

set { textBox1.Text = value.Text; }

}

public System.Windows.Forms.TextBox ResultsTextBox

{

get { return this.textBox2; }

set { textBox2.Text = value.Text; }

}

public string Heading

{

get { return this.Text; }

set { this.Text = value; }

}

//form initializer

public CommentAnalyzerForm()

{

InitializeComponent();

StaticData.mainForm = this;

this.Text += " - unnamed";

toolStripLabel1.Text = StaticData.mode;

}

//stripMenu handlers

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

{

StaticData.commands.CommandCreate();

}

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

{

StaticData.commands.CommandOpen();

}

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

{

StaticData.commands.CommandSave();

}

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

{

StaticData.commands.CommandSaveAs();

}

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

{

this.Close();

}

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

{

StaticData.commands.CommandUndo();

}

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

{

StaticData.commands.CommandRedo();

}

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

{

StaticData.commands.CommandCut();

}

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

{

StaticData.commands.CommandCopy();

}

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

{

StaticData.commands.CommandPaste();

}

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

{

StaticData.commands.CommandDelete();

}

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

{

StaticData.commands.CommandSelectAll();

}

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

{

var aboutWindow = new AboutForm();

aboutWindow.Show();

}

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

{

StaticData.commands.CommandHelp();

}

//toolstrip handlers

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

{

StaticData.commands.CommandCreate();

}

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

{

StaticData.commands.CommandOpen();

}

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

{

StaticData.commands.CommandSave();

}

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

{

StaticData.commands.CommandUndo();

}

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

{

StaticData.commands.CommandRedo();

}

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

{

StaticData.commands.CommandCopy();

}

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

{

StaticData.commands.CommandPaste();

}

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

{

StaticData.commands.CommandCut();

}

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

{

StaticData.commands.CommandHelp();

}

//textBox handlers

private void textBox1\_TextChanged(object sender, EventArgs e)

{

if (!StaticData.unsaved)

{

StaticData.unsaved = true;

this.Text += " \*";

}

}

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

var textBox = (TextBox)sender;

StaticData.undoStack.Push(textBox.Text);

}

//resize handlers

private void LanguageProcessorForm\_SizeChanged(object sender, EventArgs e)

{

splitContainer1.Width = this.Width - 20;

textBox1.Width = this.Width - 45;

textBox2.Width = this.Width - 45;

splitContainer1.Height = this.Height - 130;

}

private void splitContainer1\_Panel1\_Resize(object sender, EventArgs e)

{

textBox1.Height = splitContainer1.Panel1.Height - 7;

}

private void splitContainer1\_Panel2\_Resize(object sender, EventArgs e)

{

textBox2.Height = splitContainer1.Panel2.Height - 7;

}

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

{

StaticData.commands.CommandCheck();

}

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

{

StaticData.commands.CommandCheck();

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/task.html");

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/grammar.html");

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/classification.html");

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/analysis.html");

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/diagnostics.html");

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/tests.html");

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/books.html");

}

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

{

Help.ShowHelp(null, StaticData.prePath + "task/source.html");

}

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

{

StaticData.mode = "Automata";

toolStripLabel1.Text = StaticData.mode;

}

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

{

StaticData.mode = "ANTLR";

toolStripLabel1.Text = StaticData.mode;

}

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

{

StaticData.mode = "Flex+Bison";

toolStripLabel1.Text = StaticData.mode;

}

}

}