МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра автоматизированных систем управления



ОТЧЁТ

по РАСЧЕТНО-ГРАФИЧЕСКОМУ ЗАДАНИЮ

**«***Оператор ввода READ языка FORTRAN***»**

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

Выполнил:Проверил:

Студент гр. «*АВТ-813*», «*АВТФ*» *«д.т.н., профессор»*

*Пустовских Дмитрий*

*«Шорников Юрий Владимирович»*

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

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

(подпись) (подпись)

Новосибирск 2021

СОДЕРЖАНИЕ

[РЕФЕРАТ 3](#_Toc85807113)

[ВВЕДЕНИЕ 4](#_Toc85807114)

[1. Постановка задачи 5](#_Toc85807115)

[2. Разработка грамматики 6](#_Toc85807116)

[3. Грамматика для ANTLR 7](#_Toc85807117)

[4. Грамматика для FLEX & BISON 8](#_Toc85807118)

[5. Использование грамматики 10](#_Toc85807119)

[5.1. ANTLR 10](#_Toc85807120)

[5.2. FLEX & BISON 11](#_Toc85807121)

[6. Тестовые примеры 12](#_Toc85807122)

[7. Сравнение 17](#_Toc85807123)

[8. Листинг программы 21](#_Toc85807124)

[ЗАКЛЮЧЕНИЕ 22](#_Toc85807125)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24](#_Toc85807126)

[Приложение A 25](#_Toc85807127)

[Приложение B 38](#_Toc85807128)

# РЕФЕРАТ

Отчет 58 с., 1 кн., 17 рис., 3 источн., 2 прилож.

READ, FORTRAN, КОМПИЛЯТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, ЯЗЫКОВОЙ ПРОЦЕССОР, АВТОМАТНАЯ ГРАММАТИКА, ANTLR, FLEX & BISON.

Цель работы – с помощью программы ANTLR и Flex & Bison сгенерировать код сканера и парсера, для написанной в рамках курсовой работы грамматики, и проанализировать синтаксический разбор полученных парсеров.

В результате проектирования были рассмотрены способы генерации сканера и парсера для оператора ввода READ языка FORTRAN. Полученный код был протестирован в программе, где были получены синтаксические ошибки.

# ВВЕДЕНИЕ

Цель расчетно-графического задания – с помощью программы ANTLR и Flex & Bison сгенерировать код сканера и парсера, для написанной в рамках курсовой работы грамматики, и проанализировать синтаксический разбор полученных парсеров.

Расчетно-графическое задания содержит следующие разделы:

* Постановка задачи;
* Грамматика;
* Грамматика для ANTLR;
* Грамматика для Flex & Bison;
* Использование грамматики;
* Тестовый пример;
* Сравнение;
* Список литературы;
* Исходный код программы.

## 1. Постановка задачи

Оператор read – это оператор, который используются в программах для ввода информации в память компьютера и "считывания" значений в переменную.

Формат записи: "read(\* , <определитель формата>)<список переменных>;".

Примеры:

1. Считывание данных без определенного формата – из потока данных считываются значения без определенного формата, то есть как пользователь ввел так данные и считаются: "read(\*,\*) min;".
2. Считывание данных с заданным форматом – из потока данных считываются значения с заданным форматом, формат задается указателем на объект, который и содержит в себе формат ввода: " read (\*,100) max;"
3. Считывание множества переменных – список переменных указывается после задания формата, перечисление объектов сопровождается разделительным знаком “,”: " read(\*,\*) min,max,mean;"

Для разработанной автоматной грамматикой G[‹AB›] синтаксический анализатор (парсер) оператора read будет считать верными следующие записи:

1. "read(\*,\*) num;”
2. "read(\*,111) RC;"
3. "read(\*,\*) Num, sum, man2;"

## 2. Разработка грамматики

Определим грамматику оператора read READ языка FORTRAN G[‹AB›] в нотации Хомского с продукциями P:

1. <AB> → read <T>;
2. <T> → <O> ID{, ID }
3. <O> → (\*, <FMR>)
4. <FMR> → \* | <ЦБЗ>
5. <ID> → Б {Б | Ц}
6. <ЦБЗ> → Ц {Ц}
7. Б → a | b | c | ... | z | А | B | C | ... | Z
8. Ц → 0 | 1 | 2 | ... | 9

, где

* Б - латинские буквы верхнего и нижнего регистров;
* Ц - цифры.

Следуя введенному формальному определению грамматики, представим G[‹AB›] ее составляющими:

* Z = ‹AB›;
* VT = {a, b, c, ..., z, A, B, C, ..., Z, ( , ) , \*, ; , ., 0, 1, 2, ..., 9};
* VN = {‹AB›, ‹T›, ‹O›, ‹ FMR ›, ‹ ЦБЗ ›, <ID>}.

## 3. Грамматика для ANTLR

Из написанной грамматики несложно написать грамматику для ANTLR. Для написания грамматики необходимо расписать все правила в виде:

*neterm : expr | alternative ;*

В приведенной записи *neterm* – это нетерминальный символ, для парсера все нетерминальные символы записываются в нижнем регистре, а для сканера в верхнем регистре. *Expr* – вместо него может быть любое выражение, как терминальный символ так и нетерминальный. *Alternative* – это альтернативное выражение.

Ниже приведена реализация грамматики G[‹AB›] в ANTLR:

grammar READ;

prog : (expr NEWLINE?)+;

expr : READ term ';';

term : oper id(','id)\*;

oper : '(' '\*' ',' fmt ')';

fmt : '\*' | intg;

intg : DIGIT (DIGIT)\*;

id : SYMBOL (SYMBOL | DIGIT)\*;

READ : 'r''e''a''d';

NEWLINE : [\r?\n]+ ;

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

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

## 4. Грамматика для FLEX & BISON

Для генерации парсера с помощью FLEX & BISON необходимо реализовать 2 файла, в одном будет реализованы токены для нашей грамматики (read.l) , а в другом будут инструкции для парсера (read.y).

Ниже приведена реализация грамматики G[‹AB›] для FLEX & BISON:

**read.y:**

%{

#include <stdio.h>

%}

%token FINSENT

%token READ

%token DIGIT

%token SYMBOL

%token MULTI

%token A\_PARENT

%token C\_PARENT

%token NEWLINE

%token COMMA

%start PROG

%%

PROG: EXPR | PROG NEWLINE | ;

EXPR: READ TERM FINSENT | error;

TER: OPER VALUE | error;

VALUE: ID | VALUE COMMA VALUE | error;

OPER: A\_PARENT MULTI COMMA FMT C\_PARENT | error;

FMT: MULTI | INT | error;

INT: DIGIT| DIGIT INT | error;

ID: SYMBOL | ID DIGIT | ID SYMBOL | error;

%%

int countErr = 0;

int main (int argc, char \*\*argv)

{

extern FILE \*yyin;

yyin = fopen(argv[1], "rt");

if(yyin == NULL)

{

printf("No se puede leer el archivo seleccionado\n");

return 0;

}

yyparse();

if(countErr != 0)

printf("\nCount errors: %d\n",countErr);

else

printf("NO ERROR\n");

fclose(yyin);

}

yyerror(char \*s) {

countErr++;

extern char \*yytext;

extern int yylineno;

fprintf(stderr,"ERROR (line %d):before '%s'\n-%s\n", yylineno, yytext, s);

}

**read.l:**

%option yylineno

%option noyywrap

%option nodebug

%{

#include "readParser.h"

#include <stdio.h>

%}

DIGIT [0-9]

SYMBOL [a-zA-Z]

%%

";" { return FINSENT; }

"read" { return READ; }

"\*" { return MULTI; }

"(" { return A\_PARENT; }

")" { return C\_PARENT; }

"\n" { return NEWLINE; }

"," { return COMMA; }

{DIGIT} { return DIGIT; }

{SYMBOL} { return SYMBOL; }

%%

## 5. Использование грамматики

### 5.1. ANTLR

Для получения кода на C# необходимо в нашу грамматику добавить следующий оператор:

*options*

*{*

*language=CSharp;*

*}*

Для преобразования нашей грамматики в код необходимо обработать ее с помощью компилятора ANTLR. В результате на выходе мы получим 4 файла:

* READLexer.cs
* READParser.cs
* READLisener.cs
* READBaseLisener.cs

Для работы с полученным кодом необходимо скачать и подключить пространство имен Antlr4.Runtime. Применение сгенерированного кода в программе:

*//Создание собственного слушателя синтаксических ошибок*

*SyntaxErrorListener listener = new SyntaxErrorListener();*

*//Создание собственного слушателя синтаксических ошибок для лексера*

*SyntaxErrorListenerLexer listenerForLexer = new SyntaxErrorListenerLexer();*

*// Определение символьного потока, input – исходный текст*

*ICharStream stream = CharStreams.fromString(input);*

*// Создание лексера на потоке stream*

*READLexer lexer = new READLexer(stream);*

*// Удаление стандартных слушателей*

*lexer.RemoveErrorListeners();*

*//Добавление своего слушателя*

*lexer.AddErrorListener(**listenerForLexer);*

*//Создание потока токенов на основе лексера*

*CommonTokenStream tokens = new CommonTokenStream(lexer);*

*// Создание парсера*

*READParser parser = new READParser(tokens);*

*// Удаление стандартных слушателей*

*parser.RemoveErrorListeners();*

*//Добавление своего слушателя*

*parser.AddErrorListener(listener);*

*// Запускаем первое правило грамматики*

*parser.prog();*

В результате работы в *listener* будет находится список синтаксических ошибок для парсера, а в listenerForLexer находится список синтаксических ошибок для лексера.

Пример работы программы продемонстрирован на рисунке 1.

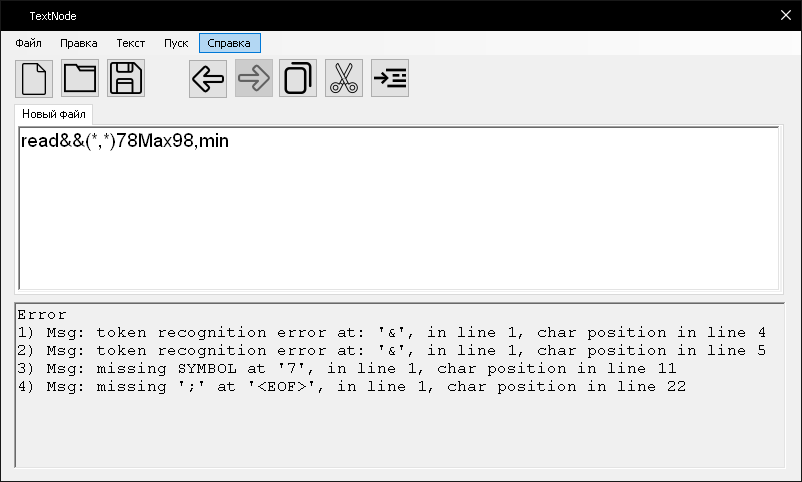


Рисунок 1 – Пример работы программы

### 5.2. FLEX & BISON

Для получения парсера и сканера необходимо реализованные файлы обработать с помощью программ flex и bison. Flex используется для разбития входного текста на токены, bison используется для генерации парсера.

В результате работы получаем сгенерированный парсер для анализа оператора read READ языка FORTRAN на C.

Пример работы программы продемонстрирован на рисунке 2

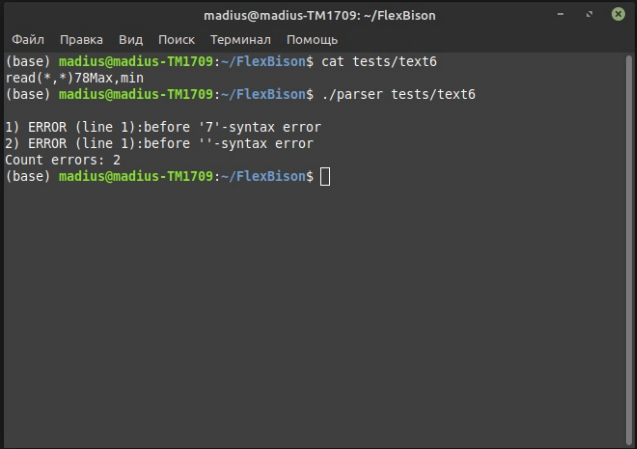


Рисунок 2 – Пример работы программы

## 6. Тестовые примеры

На рисунках 3-6 для ANTLR и на 7-10 для FLEX & BISON представлены тестовые примеры запуска сгенерированного парсера оператора ввода READ языка FORTRAN.

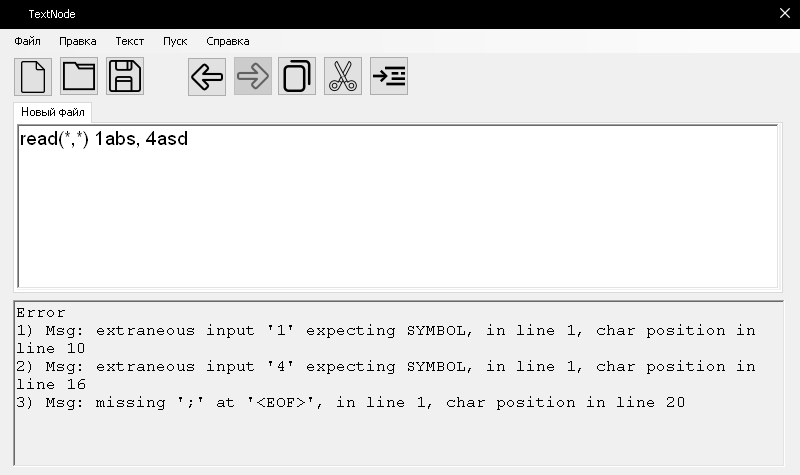


Рисунок 3 – Тестовый пример 1 для ANTLR

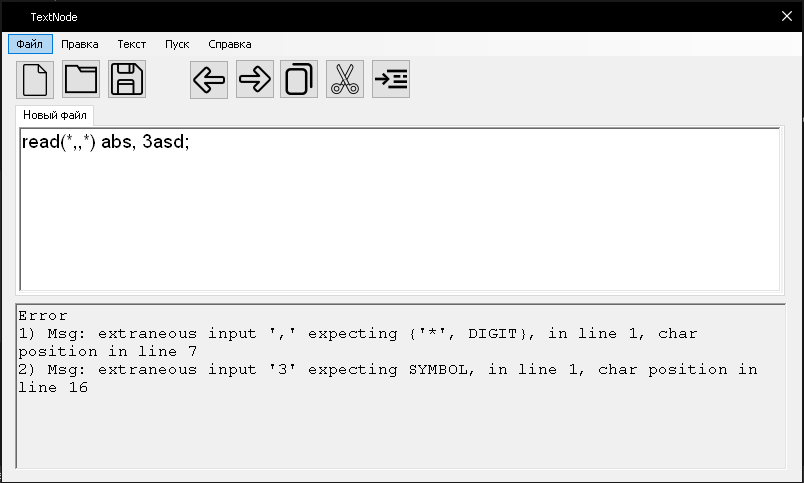


Рисунок 4 – Тестовый пример 2 для ANTLR

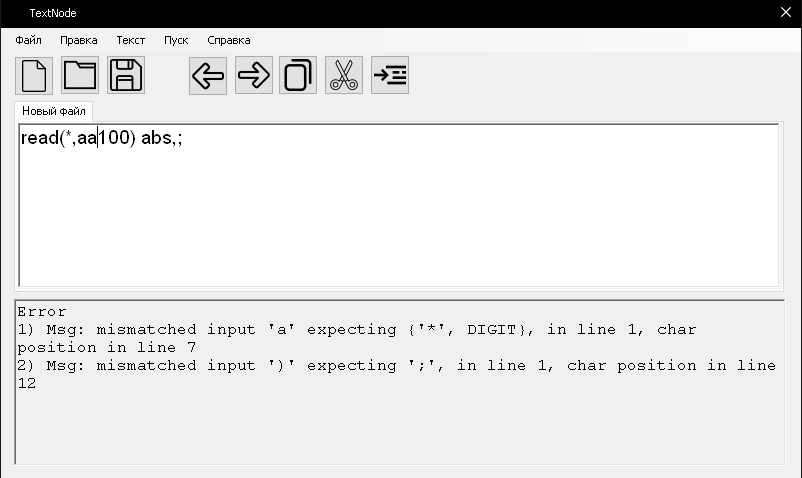


Рисунок 5 – Тестовый пример 3 для ANTLR

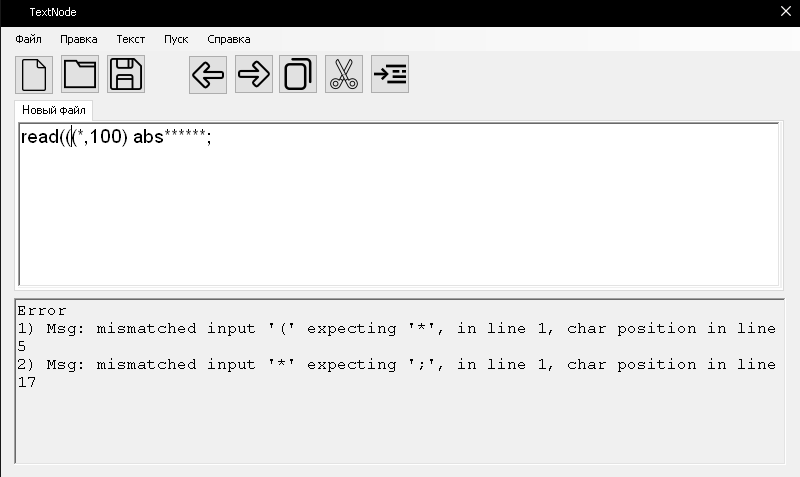


Рисунок 6 – Тестовый пример 4 для ANTLR

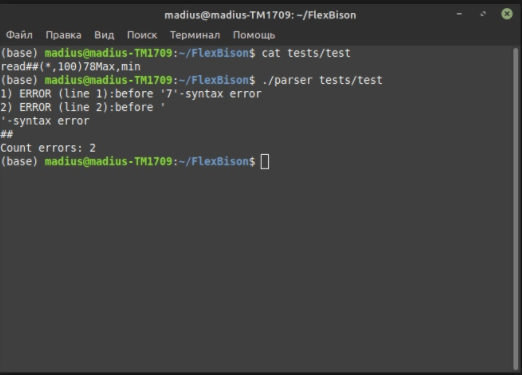


Рисунок 7 – Тестовый пример 1 для FLEX & BISON

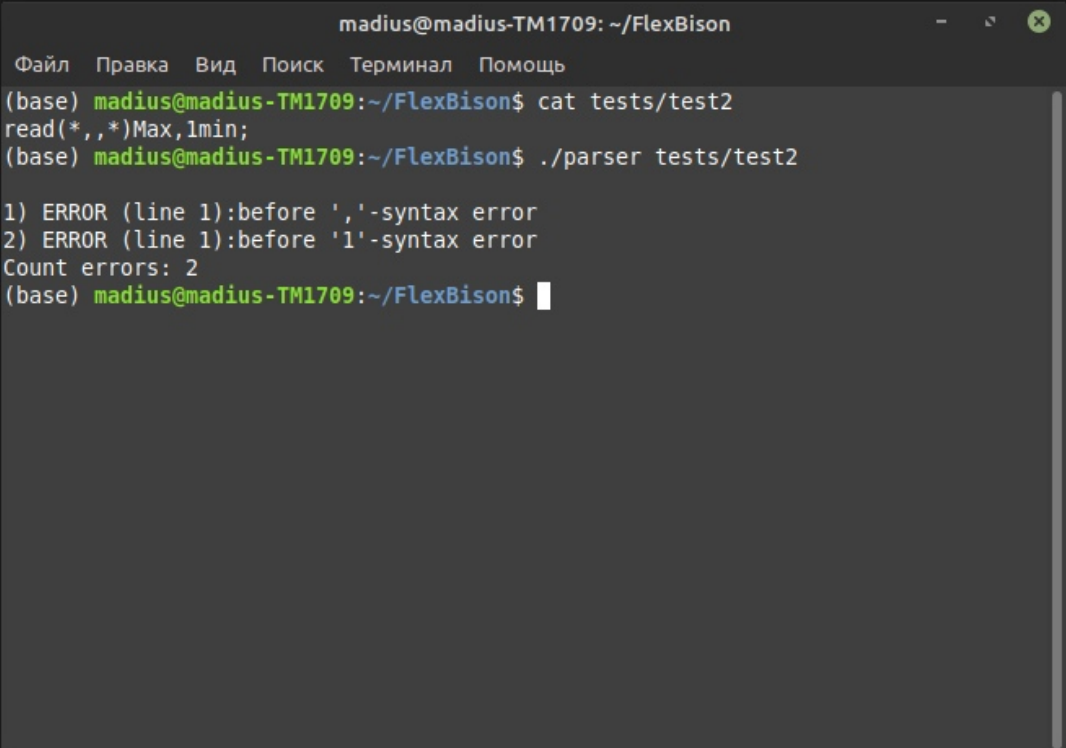


Рисунок 8 – Тестовый пример 2 для FLEX & BISON

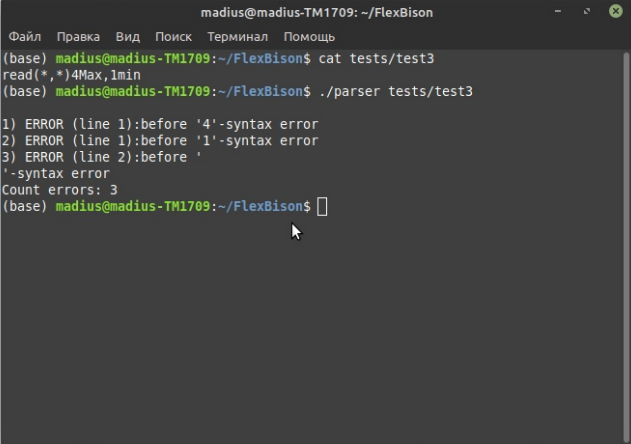


Рисунок 9 – Тестовый пример 3 для FLEX & BISON

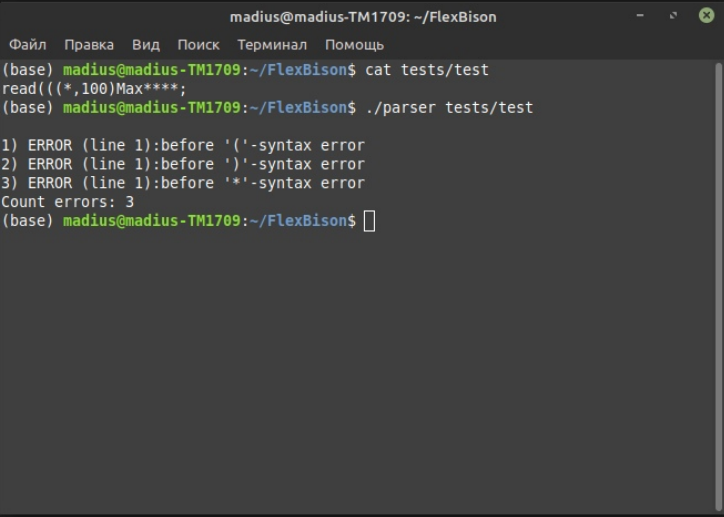


Рисунок 10 – Тестовый пример 4 для FLEX & BISON

## 7. Сравнение

Теперь сравним полученную программу с ранее разработанной в рамках курсовой работы программой, где парсер был написан с самого начала. Для примера работы возьмем те же данные, что и на рисунке 2 и 7.

На рисунке 11 показан результат вывода нашего парсера.



Рисунок 11 – Результат работы программы

Как можно увидеть наш парсер вывел 3 ошибки, у парсера сгенерированного ANTLR, было 4 ошибки. В тоже время FLEX & BISON выдал 2 ошибки, так как символы “##” были откинуты из-за того, что они не были выделены как токен. В результате получаем те же ошибки, что и у ANTLR и у наше программы.

На рисунках 12-17 представлены старые тестовые примеры парсера для оператора ввода READ языка FORTRAN и новые сообщения от парсера ANTLR и FLEX & BISON:

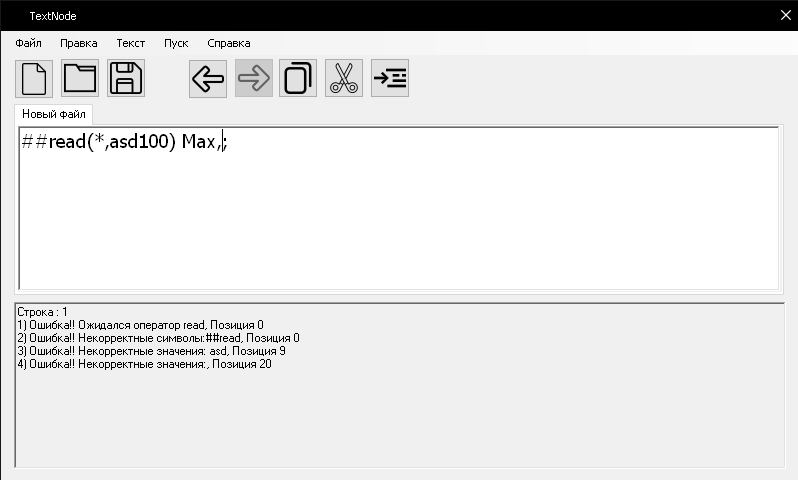


Рисунок 12 – Старый тестовый пример 2

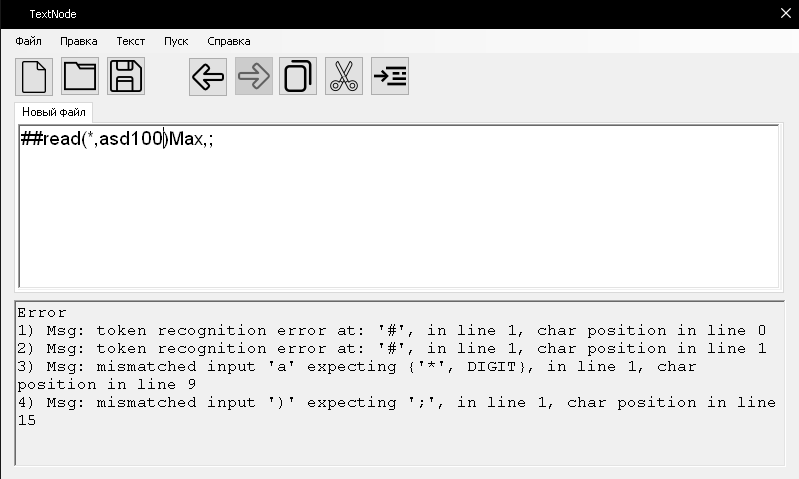


Рисунок 13 – Результат ANTLR

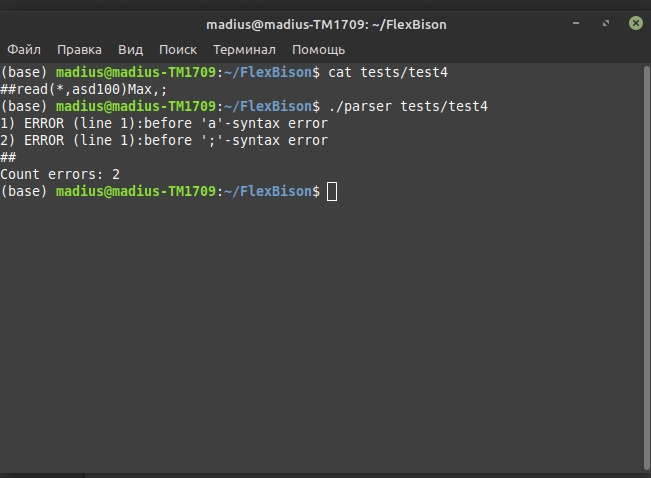


Рисунок 14 – Результат FLEX & BISON

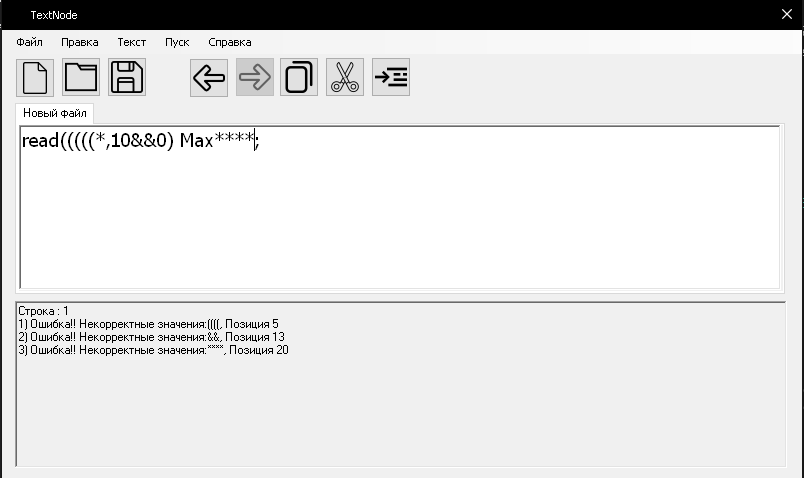


Рисунок 15 – Старый тестовый пример 3

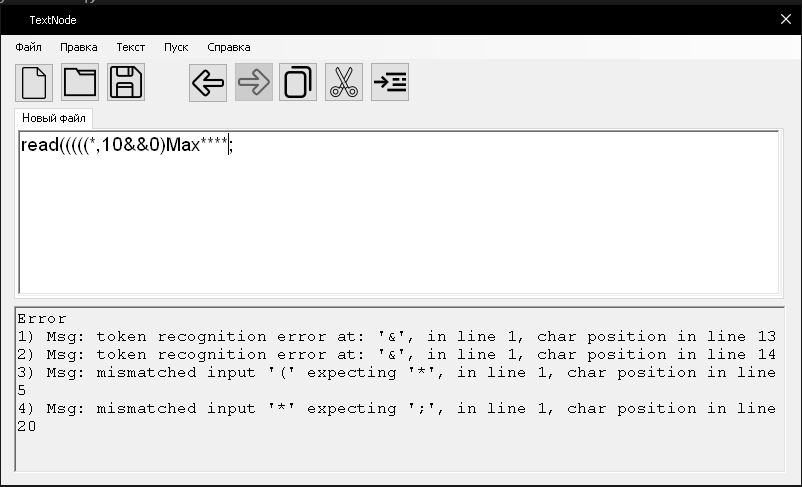


Рисунок 16 – Результат ANTLR

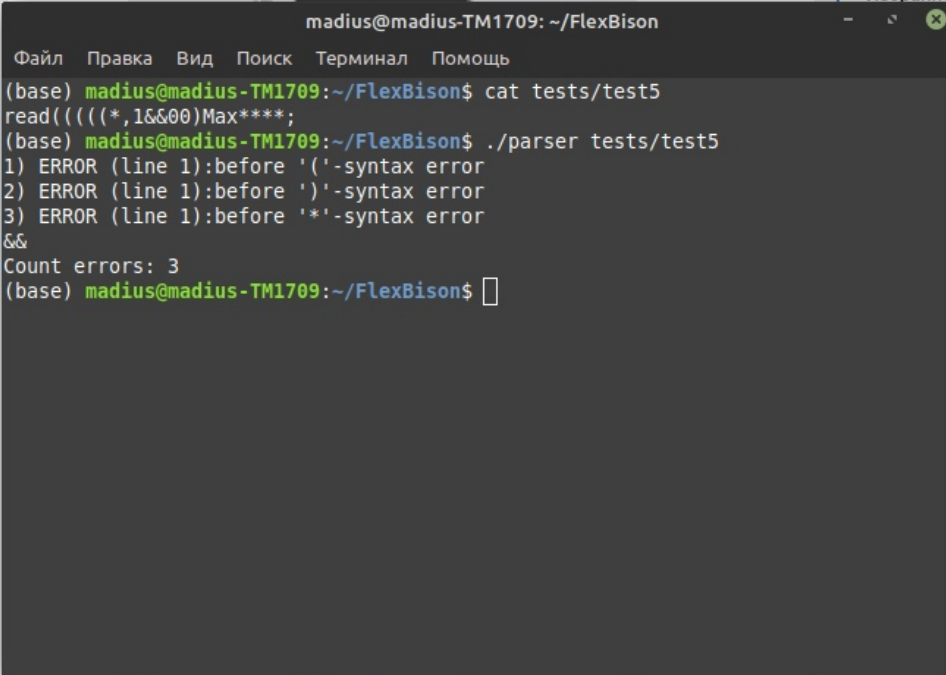


Рисунок 17 – Результат FLEX & BISON

## 8. Листинг программы

Листинг программной части сгенерированного синтаксического анализатора оператора ввода READ языка FORTRAN для ANTLR представлен в приложении А, а для FLEX & BISON в приложении B.

# ЗАКЛЮЧЕНИЕ

В результате выполнения расчетно-графического задания была написана грамматика для ANTLR. С помощью ANTLR был получен сканер и парсер на языке программирования C#. С помощью FLEX & BISON был получен парсер на языке программирования C. Протестирована работа парсера с разными входными данными и разными синтаксическими ошибками.

В результате анализа можно сделать выводы о проделанной работе. Благодаря простоте написания грамматики для конкретного генератора, можно существенно сократить разработку собственного синтаксического анализатора. Этот способ получения парсеров можно назвать безопасным так, как создание кода происходит автоматически.

Что в ANTLR и FLEX & BISON анализатор работает хорошо, но при построении синтаксического дерева не может в некоторых случаях достаивать его для продолжения анализа ошибок. В результате этого многие ошибки могут быть не распознаны. В реализованном нами парсере такие ошибки мы учитываем и строим дерево так, чтобы распознать как можно больше ошибок.

ANTLR в диагностике ошибок работает лучше чем FLEX & BISON. Многие вещи в FLEX & BISON не позволяют настроить под конкретные задачи, в ту же очередь ANTLR позволяет настроить парсер так как того требует задача.

В результате работы можно сделать вывод о том, что ANTLR и FLEX & BISON, значительно упрощают процесс создания собственного синтаксического анализатора. Они идеально подходят для создания парсеров с большим количеством продукций в грамматике. Но в тоже время автоматическая генерация не позволяет провести полную диагностику. Поэтому для выявления всех ошибок необходимо реализовывать парсер своими руками.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шорников Ю.В. Теория и практика языковых процессоров: учеб. пособие / Ю.В. Шорников. – Новосибирск: Изд-во НГТУ, 2004.
2. ANTLR [Электронный ресурс] / Электрон. дан. URL: https://www.antlr.org/, свободный. Анг.рус. (дата обращения 13.10.2021).
3. CSharp-target [Электронный ресурс] / Электрон. дан. URL: https://github.com/antlr/antlr4/blob/master/doc/csharp-target.md, свободный. Анг.рус. (дата обращения 13.10.2021).

# Приложение A

READParser.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 READ.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.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 READParser : Parser {

protected static DFA[] decisionToDFA;

protected static PredictionContextCache sharedContextCache = new PredictionContextCache();

public const int

T\_\_0=1, T\_\_1=2, T\_\_2=3, T\_\_3=4, T\_\_4=5, READ=6, NEWLINE=7, SYMBOL=8, DIGIT=9;

public const int

RULE\_prog = 0, RULE\_expr = 1, RULE\_term = 2, RULE\_oper = 3, RULE\_fmt = 4,

RULE\_intg = 5, RULE\_id = 6;

public static readonly string[] ruleNames = {

"prog", "expr", "term", "oper", "fmt", "intg", "id"

};

private static readonly string[] \_LiteralNames = {

null, "';'", "','", "'('", "'\*'", "')'"

};

private static readonly string[] \_SymbolicNames = {

null, null, null, null, null, null, "READ", "NEWLINE", "SYMBOL", "DIGIT"

};

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

[NotNull]

public override IVocabulary Vocabulary

{

get

{

return DefaultVocabulary;

}

}

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

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

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

static READParser() {

decisionToDFA = new DFA[\_ATN.NumberOfDecisions];

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

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

}

}

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

public READParser(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 ExprContext[] expr() {

return GetRuleContexts<ExprContext>();

}

[System.Diagnostics.DebuggerNonUserCode] public ExprContext expr(int i) {

return GetRuleContext<ExprContext>(i);

}

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode[] NEWLINE() { return GetTokens(READParser.NEWLINE); }

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

return GetToken(READParser.NEWLINE, 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) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override TResult Accept<TResult>(IParseTreeVisitor<TResult> visitor) {

IREADVisitor<TResult> typedVisitor = visitor as IREADVisitor<TResult>;

if (typedVisitor != null) return typedVisitor.VisitProg(this);

else return visitor.VisitChildren(this);

}

}

[RuleVersion(0)]

public ProgContext prog() {

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

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

int \_la;

try {

EnterOuterAlt(\_localctx, 1);

{

State = 18;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

do {

{

{

State = 14;

expr();

State = 16;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

if (\_la==NEWLINE) {

{

State = 15;

Match(NEWLINE);

}

}

}

}

State = 20;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

} while ( \_la==READ );

}

}

catch (RecognitionException re) {

\_localctx.exception = re;

ErrorHandler.ReportError(this, re);

ErrorHandler.Recover(this, re);

}

finally {

ExitRule();

}

return \_localctx;

}

public partial class ExprContext : ParserRuleContext {

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

[System.Diagnostics.DebuggerNonUserCode] public TermContext term() {

return GetRuleContext<TermContext>(0);

}

public ExprContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

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

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override TResult Accept<TResult>(IParseTreeVisitor<TResult> visitor) {

IREADVisitor<TResult> typedVisitor = visitor as IREADVisitor<TResult>;

if (typedVisitor != null) return typedVisitor.VisitExpr(this);

else return visitor.VisitChildren(this);

}

}

[RuleVersion(0)]

public ExprContext expr() {

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

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

try {

EnterOuterAlt(\_localctx, 1);

{

State = 22;

Match(READ);

State = 23;

term();

State = 24;

Match(T\_\_0);

}

}

catch (RecognitionException re) {

\_localctx.exception = re;

ErrorHandler.ReportError(this, re);

ErrorHandler.Recover(this, re);

}

finally {

ExitRule();

}

return \_localctx;

}

public partial class TermContext : ParserRuleContext {

[System.Diagnostics.DebuggerNonUserCode] public OperContext oper() {

return GetRuleContext<OperContext>(0);

}

[System.Diagnostics.DebuggerNonUserCode] public IdContext[] id() {

return GetRuleContexts<IdContext>();

}

[System.Diagnostics.DebuggerNonUserCode] public IdContext id(int i) {

return GetRuleContext<IdContext>(i);

}

public TermContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

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

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override TResult Accept<TResult>(IParseTreeVisitor<TResult> visitor) {

IREADVisitor<TResult> typedVisitor = visitor as IREADVisitor<TResult>;

if (typedVisitor != null) return typedVisitor.VisitTerm(this);

else return visitor.VisitChildren(this);

}

}

[RuleVersion(0)]

public TermContext term() {

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

EnterRule(\_localctx, 4, RULE\_term);

int \_la;

try {

EnterOuterAlt(\_localctx, 1);

{

State = 26;

oper();

State = 27;

id();

State = 32;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

while (\_la==T\_\_1) {

{

{

State = 28;

Match(T\_\_1);

State = 29;

id();

}

}

State = 34;

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;

}

public partial class OperContext : ParserRuleContext {

[System.Diagnostics.DebuggerNonUserCode] public FmtContext fmt() {

return GetRuleContext<FmtContext>(0);

}

public OperContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

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

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override TResult Accept<TResult>(IParseTreeVisitor<TResult> visitor) {

IREADVisitor<TResult> typedVisitor = visitor as IREADVisitor<TResult>;

if (typedVisitor != null) return typedVisitor.VisitOper(this);

else return visitor.VisitChildren(this);

}

}

[RuleVersion(0)]

public OperContext oper() {

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

EnterRule(\_localctx, 6, RULE\_oper);

try {

EnterOuterAlt(\_localctx, 1);

{

State = 35;

Match(T\_\_2);

State = 36;

Match(T\_\_3);

State = 37;

Match(T\_\_1);

State = 38;

fmt();

State = 39;

Match(T\_\_4);

}

}

catch (RecognitionException re) {

\_localctx.exception = re;

ErrorHandler.ReportError(this, re);

ErrorHandler.Recover(this, re);

}

finally {

ExitRule();

}

return \_localctx;

}

public partial class FmtContext : ParserRuleContext {

[System.Diagnostics.DebuggerNonUserCode] public IntgContext intg() {

return GetRuleContext<IntgContext>(0);

}

public FmtContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

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

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override TResult Accept<TResult>(IParseTreeVisitor<TResult> visitor) {

IREADVisitor<TResult> typedVisitor = visitor as IREADVisitor<TResult>;

if (typedVisitor != null) return typedVisitor.VisitFmt(this);

else return visitor.VisitChildren(this);

}

}

[RuleVersion(0)]

public FmtContext fmt() {

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

EnterRule(\_localctx, 8, RULE\_fmt);

try {

State = 43;

ErrorHandler.Sync(this);

switch (TokenStream.LA(1)) {

case T\_\_3:

EnterOuterAlt(\_localctx, 1);

{

State = 41;

Match(T\_\_3);

}

break;

case DIGIT:

EnterOuterAlt(\_localctx, 2);

{

State = 42;

intg();

}

break;

default:

throw new NoViableAltException(this);

}

}

catch (RecognitionException re) {

\_localctx.exception = re;

ErrorHandler.ReportError(this, re);

ErrorHandler.Recover(this, re);

}

finally {

ExitRule();

}

return \_localctx;

}

public partial class IntgContext : ParserRuleContext {

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

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

return GetToken(READParser.DIGIT, i);

}

public IntgContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

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

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override TResult Accept<TResult>(IParseTreeVisitor<TResult> visitor) {

IREADVisitor<TResult> typedVisitor = visitor as IREADVisitor<TResult>;

if (typedVisitor != null) return typedVisitor.VisitIntg(this);

else return visitor.VisitChildren(this);

}

}

[RuleVersion(0)]

public IntgContext intg() {

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

EnterRule(\_localctx, 10, RULE\_intg);

int \_la;

try {

EnterOuterAlt(\_localctx, 1);

{

State = 45;

Match(DIGIT);

State = 49;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

while (\_la==DIGIT) {

{

{

State = 46;

Match(DIGIT);

}

}

State = 51;

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;

}

public partial class IdContext : ParserRuleContext {

[System.Diagnostics.DebuggerNonUserCode] public ITerminalNode[] SYMBOL() { return GetTokens(READParser.SYMBOL); }

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

return GetToken(READParser.SYMBOL, i);

}

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

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

return GetToken(READParser.DIGIT, i);

}

public IdContext(ParserRuleContext parent, int invokingState)

: base(parent, invokingState)

{

}

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

[System.Diagnostics.DebuggerNonUserCode]

public override void EnterRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override void ExitRule(IParseTreeListener listener) {

IREADListener typedListener = listener as IREADListener;

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

}

[System.Diagnostics.DebuggerNonUserCode]

public override TResult Accept<TResult>(IParseTreeVisitor<TResult> visitor) {

IREADVisitor<TResult> typedVisitor = visitor as IREADVisitor<TResult>;

if (typedVisitor != null) return typedVisitor.VisitId(this);

else return visitor.VisitChildren(this);

}

}

[RuleVersion(0)]

public IdContext id() {

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

EnterRule(\_localctx, 12, RULE\_id);

int \_la;

try {

EnterOuterAlt(\_localctx, 1);

{

State = 52;

Match(SYMBOL);

State = 56;

ErrorHandler.Sync(this);

\_la = TokenStream.LA(1);

while (\_la==SYMBOL || \_la==DIGIT) {

{

{

State = 53;

\_la = TokenStream.LA(1);

if ( !(\_la==SYMBOL || \_la==DIGIT) ) {

ErrorHandler.RecoverInline(this);

}

else {

ErrorHandler.ReportMatch(this);

Consume();

}

}

}

State = 58;

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', '\v', '>', '\x4', '\x2', '\t', '\x2', '\x4', '\x3', '\t',

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

'\t', '\x6', '\x4', '\a', '\t', '\a', '\x4', '\b', '\t', '\b', '\x3',

'\x2', '\x3', '\x2', '\x5', '\x2', '\x13', '\n', '\x2', '\x6', '\x2',

'\x15', '\n', '\x2', '\r', '\x2', '\xE', '\x2', '\x16', '\x3', '\x3',

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

'\x3', '\x4', '\x3', '\x4', '\a', '\x4', '!', '\n', '\x4', '\f', '\x4',

'\xE', '\x4', '$', '\v', '\x4', '\x3', '\x5', '\x3', '\x5', '\x3', '\x5',

'\x3', '\x5', '\x3', '\x5', '\x3', '\x5', '\x3', '\x6', '\x3', '\x6',

'\x5', '\x6', '.', '\n', '\x6', '\x3', '\a', '\x3', '\a', '\a', '\a',

'\x32', '\n', '\a', '\f', '\a', '\xE', '\a', '\x35', '\v', '\a', '\x3',

'\b', '\x3', '\b', '\a', '\b', '\x39', '\n', '\b', '\f', '\b', '\xE',

'\b', '<', '\v', '\b', '\x3', '\b', '\x2', '\x2', '\t', '\x2', '\x4',

'\x6', '\b', '\n', '\f', '\xE', '\x2', '\x3', '\x3', '\x2', '\n', '\v',

'\x2', '<', '\x2', '\x14', '\x3', '\x2', '\x2', '\x2', '\x4', '\x18',

'\x3', '\x2', '\x2', '\x2', '\x6', '\x1C', '\x3', '\x2', '\x2', '\x2',

'\b', '%', '\x3', '\x2', '\x2', '\x2', '\n', '-', '\x3', '\x2', '\x2',

'\x2', '\f', '/', '\x3', '\x2', '\x2', '\x2', '\xE', '\x36', '\x3', '\x2',

'\x2', '\x2', '\x10', '\x12', '\x5', '\x4', '\x3', '\x2', '\x11', '\x13',

'\a', '\t', '\x2', '\x2', '\x12', '\x11', '\x3', '\x2', '\x2', '\x2',

'\x12', '\x13', '\x3', '\x2', '\x2', '\x2', '\x13', '\x15', '\x3', '\x2',

'\x2', '\x2', '\x14', '\x10', '\x3', '\x2', '\x2', '\x2', '\x15', '\x16',

'\x3', '\x2', '\x2', '\x2', '\x16', '\x14', '\x3', '\x2', '\x2', '\x2',

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

'\x2', '\x2', '\x18', '\x19', '\a', '\b', '\x2', '\x2', '\x19', '\x1A',

'\x5', '\x6', '\x4', '\x2', '\x1A', '\x1B', '\a', '\x3', '\x2', '\x2',

'\x1B', '\x5', '\x3', '\x2', '\x2', '\x2', '\x1C', '\x1D', '\x5', '\b',

'\x5', '\x2', '\x1D', '\"', '\x5', '\xE', '\b', '\x2', '\x1E', '\x1F',

'\a', '\x4', '\x2', '\x2', '\x1F', '!', '\x5', '\xE', '\b', '\x2', ' ',

'\x1E', '\x3', '\x2', '\x2', '\x2', '!', '$', '\x3', '\x2', '\x2', '\x2',

'\"', ' ', '\x3', '\x2', '\x2', '\x2', '\"', '#', '\x3', '\x2', '\x2',

'\x2', '#', '\a', '\x3', '\x2', '\x2', '\x2', '$', '\"', '\x3', '\x2',

'\x2', '\x2', '%', '&', '\a', '\x5', '\x2', '\x2', '&', '\'', '\a', '\x6',

'\x2', '\x2', '\'', '(', '\a', '\x4', '\x2', '\x2', '(', ')', '\x5', '\n',

'\x6', '\x2', ')', '\*', '\a', '\a', '\x2', '\x2', '\*', '\t', '\x3', '\x2',

'\x2', '\x2', '+', '.', '\a', '\x6', '\x2', '\x2', ',', '.', '\x5', '\f',

'\a', '\x2', '-', '+', '\x3', '\x2', '\x2', '\x2', '-', ',', '\x3', '\x2',

'\x2', '\x2', '.', '\v', '\x3', '\x2', '\x2', '\x2', '/', '\x33', '\a',

'\v', '\x2', '\x2', '\x30', '\x32', '\a', '\v', '\x2', '\x2', '\x31',

'\x30', '\x3', '\x2', '\x2', '\x2', '\x32', '\x35', '\x3', '\x2', '\x2',

'\x2', '\x33', '\x31', '\x3', '\x2', '\x2', '\x2', '\x33', '\x34', '\x3',

'\x2', '\x2', '\x2', '\x34', '\r', '\x3', '\x2', '\x2', '\x2', '\x35',

'\x33', '\x3', '\x2', '\x2', '\x2', '\x36', ':', '\a', '\n', '\x2', '\x2',

'\x37', '\x39', '\t', '\x2', '\x2', '\x2', '\x38', '\x37', '\x3', '\x2',

'\x2', '\x2', '\x39', '<', '\x3', '\x2', '\x2', '\x2', ':', '\x38', '\x3',

'\x2', '\x2', '\x2', ':', ';', '\x3', '\x2', '\x2', '\x2', ';', '\xF',

'\x3', '\x2', '\x2', '\x2', '<', ':', '\x3', '\x2', '\x2', '\x2', '\b',

'\x12', '\x16', '\"', '-', '\x33', ':',

};

public static readonly ATN \_ATN =

new ATNDeserializer().Deserialize(\_serializedATN);

}

READLexer.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 READ.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.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 READLexer : Lexer {

protected static DFA[] decisionToDFA;

protected static PredictionContextCache sharedContextCache = new PredictionContextCache();

public const int

T\_\_0=1, T\_\_1=2, T\_\_2=3, T\_\_3=4, T\_\_4=5, READ=6, NEWLINE=7, SYMBOL=8, DIGIT=9;

public static string[] channelNames = {

"DEFAULT\_TOKEN\_CHANNEL", "HIDDEN"

};

public static string[] modeNames = {

"DEFAULT\_MODE"

};

public static readonly string[] ruleNames = {

"T\_\_0", "T\_\_1", "T\_\_2", "T\_\_3", "T\_\_4", "READ", "NEWLINE", "SYMBOL", "DIGIT"

};

public READLexer(ICharStream input)

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

public READLexer(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, null, null, null, null, null, "READ", "NEWLINE", "SYMBOL", "DIGIT"

};

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

[NotNull]

public override IVocabulary Vocabulary

{

get

{

return DefaultVocabulary;

}

}

public override string GrammarFileName { get { return "READ.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 READLexer() {

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', '\v', '-', '\b', '\x1', '\x4', '\x2', '\t', '\x2', '\x4',

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

'\x4', '\x6', '\t', '\x6', '\x4', '\a', '\t', '\a', '\x4', '\b', '\t',

'\b', '\x4', '\t', '\t', '\t', '\x4', '\n', '\t', '\n', '\x3', '\x2',

'\x3', '\x2', '\x3', '\x3', '\x3', '\x3', '\x3', '\x4', '\x3', '\x4',

'\x3', '\x5', '\x3', '\x5', '\x3', '\x6', '\x3', '\x6', '\x3', '\a', '\x3',

'\a', '\x3', '\a', '\x3', '\a', '\x3', '\a', '\x3', '\b', '\x6', '\b',

'&', '\n', '\b', '\r', '\b', '\xE', '\b', '\'', '\x3', '\t', '\x3', '\t',

'\x3', '\n', '\x3', '\n', '\x2', '\x2', '\v', '\x3', '\x3', '\x5', '\x4',

'\a', '\x5', '\t', '\x6', '\v', '\a', '\r', '\b', '\xF', '\t', '\x11',

'\n', '\x13', '\v', '\x3', '\x2', '\x4', '\x4', '\x2', '\f', '\f', '\xF',

'\xF', '\x4', '\x2', '\x43', '\\', '\x63', '|', '\x2', '-', '\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', '\x2', '\r', '\x3', '\x2',

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

'\x3', '\x2', '\x2', '\x2', '\x2', '\x13', '\x3', '\x2', '\x2', '\x2',

'\x3', '\x15', '\x3', '\x2', '\x2', '\x2', '\x5', '\x17', '\x3', '\x2',

'\x2', '\x2', '\a', '\x19', '\x3', '\x2', '\x2', '\x2', '\t', '\x1B',

'\x3', '\x2', '\x2', '\x2', '\v', '\x1D', '\x3', '\x2', '\x2', '\x2',

'\r', '\x1F', '\x3', '\x2', '\x2', '\x2', '\xF', '%', '\x3', '\x2', '\x2',

'\x2', '\x11', ')', '\x3', '\x2', '\x2', '\x2', '\x13', '+', '\x3', '\x2',

'\x2', '\x2', '\x15', '\x16', '\a', '=', '\x2', '\x2', '\x16', '\x4',

'\x3', '\x2', '\x2', '\x2', '\x17', '\x18', '\a', '.', '\x2', '\x2', '\x18',

'\x6', '\x3', '\x2', '\x2', '\x2', '\x19', '\x1A', '\a', '\*', '\x2', '\x2',

'\x1A', '\b', '\x3', '\x2', '\x2', '\x2', '\x1B', '\x1C', '\a', ',', '\x2',

'\x2', '\x1C', '\n', '\x3', '\x2', '\x2', '\x2', '\x1D', '\x1E', '\a',

'+', '\x2', '\x2', '\x1E', '\f', '\x3', '\x2', '\x2', '\x2', '\x1F', ' ',

'\a', 't', '\x2', '\x2', ' ', '!', '\a', 'g', '\x2', '\x2', '!', '\"',

'\a', '\x63', '\x2', '\x2', '\"', '#', '\a', '\x66', '\x2', '\x2', '#',

'\xE', '\x3', '\x2', '\x2', '\x2', '$', '&', '\t', '\x2', '\x2', '\x2',

'%', '$', '\x3', '\x2', '\x2', '\x2', '&', '\'', '\x3', '\x2', '\x2',

'\x2', '\'', '%', '\x3', '\x2', '\x2', '\x2', '\'', '(', '\x3', '\x2',

'\x2', '\x2', '(', '\x10', '\x3', '\x2', '\x2', '\x2', ')', '\*', '\t',

'\x3', '\x2', '\x2', '\*', '\x12', '\x3', '\x2', '\x2', '\x2', '+', ',',

'\x4', '\x32', ';', '\x2', ',', '\x14', '\x3', '\x2', '\x2', '\x2', '\x4',

'\x2', '\'', '\x2',

};

public static readonly ATN \_ATN =

new ATNDeserializer().Deserialize(\_serializedATN);

}

ErrorListener.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Antlr4.Runtime;

namespace TFLK\_lab\_1\_

{

public class SyntaxError

{

public TextWriter output;

public IRecognizer recognizer;

public IToken offendingSymbol;

public int line;

public int charPositionInLine;

public string msg;

public RecognitionException e;

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

{

this.output = output;

this.recognizer = recognizer;

this.offendingSymbol = offendingSymbol;

this.line = line;

this.charPositionInLine = charPositionInLine;

this.e = e;

this.msg = msg;

}

public override string ToString()

{

string str = "Msg: ";

str += msg;

str += ", in line " + line;

str += ", char position in line " + charPositionInLine;

return str;

}

}

public class SyntaxErrorListener : BaseErrorListener

{

List<SyntaxError> syntaxErrors;

public List<SyntaxError> SyntaxErrors { get => syntaxErrors;}

public SyntaxErrorListener()

{

syntaxErrors = new List<SyntaxError>();

}

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

{

syntaxErrors.Add(new SyntaxError(output, recognizer, offendingSymbol, line, charPositionInLine, msg, e));

}

}

public class SyntaxErrorListenerLexer : IAntlrErrorListener<int>

{

List<SyntaxError> syntaxErrors;

public List<SyntaxError> SyntaxErrors { get => syntaxErrors; }

public SyntaxErrorListenerLexer()

{

syntaxErrors = new List<SyntaxError>();

}

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

{

syntaxErrors.Add(new SyntaxError(output, recognizer, null, line, charPositionInLine, msg, e));

}

}

}

# Приложение B

/\* A Bison parser, made by GNU Bison 3.5.1. \*/

/\* Bison implementation for Yacc-like parsers in C

Copyright (C) 1984, 1989-1990, 2000-2015, 2018-2020 Free Software Foundation,

Inc.

This program is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program. If not, see <http://www.gnu.org/licenses/>. \*/

/\* As a special exception, you may create a larger work that contains

part or all of the Bison parser skeleton and distribute that work

under terms of your choice, so long as that work isn't itself a

parser generator using the skeleton or a modified version thereof

as a parser skeleton. Alternatively, if you modify or redistribute

the parser skeleton itself, you may (at your option) remove this

special exception, which will cause the skeleton and the resulting

Bison output files to be licensed under the GNU General Public

License without this special exception.

This special exception was added by the Free Software Foundation in

version 2.2 of Bison. \*/

/\* C LALR(1) parser skeleton written by Richard Stallman, by

simplifying the original so-called "semantic" parser. \*/

/\* All symbols defined below should begin with yy or YY, to avoid

infringing on user name space. This should be done even for local

variables, as they might otherwise be expanded by user macros.

There are some unavoidable exceptions within include files to

define necessary library symbols; they are noted "INFRINGES ON

USER NAME SPACE" below. \*/

/\* Undocumented macros, especially those whose name start with YY\_,

are private implementation details. Do not rely on them. \*/

/\* Identify Bison output. \*/

#define YYBISON 1

/\* Bison version. \*/

#define YYBISON\_VERSION "3.5.1"

/\* Skeleton name. \*/

#define YYSKELETON\_NAME "yacc.c"

/\* Pure parsers. \*/

#define YYPURE 0

/\* Push parsers. \*/

#define YYPUSH 0

/\* Pull parsers. \*/

#define YYPULL 1

/\* First part of user prologue. \*/

#line 1 "GRAM.y"

#include <stdio.h>

#include <string.h>

#line 75 "readParser.c"

# ifndef YY\_CAST

# ifdef \_\_cplusplus

# define YY\_CAST(Type, Val) static\_cast<Type> (Val)

# define YY\_REINTERPRET\_CAST(Type, Val) reinterpret\_cast<Type> (Val)

# else

# define YY\_CAST(Type, Val) ((Type) (Val))

# define YY\_REINTERPRET\_CAST(Type, Val) ((Type) (Val))

# endif

# endif

# ifndef YY\_NULLPTR

# if defined \_\_cplusplus

# if 201103L <= \_\_cplusplus

# define YY\_NULLPTR nullptr

# else

# define YY\_NULLPTR 0

# endif

# else

# define YY\_NULLPTR ((void\*)0)

# endif

# endif

/\* Enabling verbose error messages. \*/

#ifdef YYERROR\_VERBOSE

# undef YYERROR\_VERBOSE

# define YYERROR\_VERBOSE 1

#else

# define YYERROR\_VERBOSE 0

#endif

/\* Use api.header.include to #include this header

instead of duplicating it here. \*/

#ifndef YY\_YY\_READPARSER\_H\_INCLUDED

# define YY\_YY\_READPARSER\_H\_INCLUDED

/\* Debug traces. \*/

#ifndef YYDEBUG

# define YYDEBUG 0

#endif

#if YYDEBUG

extern int yydebug;

#endif

/\* Token type. \*/

#ifndef YYTOKENTYPE

# define YYTOKENTYPE

enum **yytokentype**

{

FINSENT = 258,

READ = 259,

DIGIT = 260,

SYMBOL = 261,

MULTI = 262,

A\_PARENT = 263,

C\_PARENT = 264,

NEWLINE = 265,

COMMA = 266

};

#endif

/\* Value type. \*/

#if ! defined YYSTYPE && ! defined YYSTYPE\_IS\_DECLARED

typedef int YYSTYPE;

# define YYSTYPE\_IS\_TRIVIAL 1

# define YYSTYPE\_IS\_DECLARED 1

#endif

extern YYSTYPE yylval;

int **yyparse** (void);

#endif /\* !YY\_YY\_READPARSER\_H\_INCLUDED \*/

#ifdef short

# undef short

#endif

/\* On compilers that do not define \_\_PTRDIFF\_MAX\_\_ etc., make sure

<limits.h> and (if available) <stdint.h> are included

so that the code can choose integer types of a good width. \*/

#ifndef \_\_PTRDIFF\_MAX\_\_

# include <limits.h> /\* INFRINGES ON USER NAME SPACE \*/

# if defined \_\_STDC\_VERSION\_\_ && 199901 <= \_\_STDC\_VERSION\_\_

# include <stdint.h> /\* INFRINGES ON USER NAME SPACE \*/

# define YY\_STDINT\_H

# endif

#endif

/\* Narrow types that promote to a signed type and that can represent a

signed or unsigned integer of at least N bits. In tables they can

save space and decrease cache pressure. Promoting to a signed type

helps avoid bugs in integer arithmetic. \*/

#ifdef \_\_INT\_LEAST8\_MAX\_\_

typedef \_\_INT\_LEAST8\_TYPE\_\_ yytype\_int8;

#elif defined YY\_STDINT\_H

typedef int\_least8\_t yytype\_int8;

#else

typedef signed char yytype\_int8;

#endif

#ifdef \_\_INT\_LEAST16\_MAX\_\_

typedef \_\_INT\_LEAST16\_TYPE\_\_ yytype\_int16;

#elif defined YY\_STDINT\_H

typedef int\_least16\_t yytype\_int16;

#else

typedef short yytype\_int16;

#endif

#if defined \_\_UINT\_LEAST8\_MAX\_\_ && \_\_UINT\_LEAST8\_MAX\_\_ <= \_\_INT\_MAX\_\_

typedef \_\_UINT\_LEAST8\_TYPE\_\_ yytype\_uint8;

#elif (!defined \_\_UINT\_LEAST8\_MAX\_\_ && defined YY\_STDINT\_H \

&& UINT\_LEAST8\_MAX <= INT\_MAX)

typedef uint\_least8\_t yytype\_uint8;

#elif !defined \_\_UINT\_LEAST8\_MAX\_\_ && UCHAR\_MAX <= INT\_MAX

typedef unsigned char yytype\_uint8;

#else

typedef short yytype\_uint8;

#endif

#if defined \_\_UINT\_LEAST16\_MAX\_\_ && \_\_UINT\_LEAST16\_MAX\_\_ <= \_\_INT\_MAX\_\_

typedef \_\_UINT\_LEAST16\_TYPE\_\_ yytype\_uint16;

#elif (!defined \_\_UINT\_LEAST16\_MAX\_\_ && defined YY\_STDINT\_H \

&& UINT\_LEAST16\_MAX <= INT\_MAX)

typedef uint\_least16\_t yytype\_uint16;

#elif !defined \_\_UINT\_LEAST16\_MAX\_\_ && USHRT\_MAX <= INT\_MAX

typedef unsigned short yytype\_uint16;

#else

typedef int yytype\_uint16;

#endif

#ifndef YYPTRDIFF\_T

# if defined \_\_PTRDIFF\_TYPE\_\_ && defined \_\_PTRDIFF\_MAX\_\_

# define YYPTRDIFF\_T \_\_PTRDIFF\_TYPE\_\_

# define YYPTRDIFF\_MAXIMUM \_\_PTRDIFF\_MAX\_\_

# elif defined PTRDIFF\_MAX

# ifndef ptrdiff\_t

# include <stddef.h> /\* INFRINGES ON USER NAME SPACE \*/

# endif

# define YYPTRDIFF\_T ptrdiff\_t

# define YYPTRDIFF\_MAXIMUM PTRDIFF\_MAX

# else

# define YYPTRDIFF\_T long

# define YYPTRDIFF\_MAXIMUM LONG\_MAX

# endif

#endif

#ifndef YYSIZE\_T

# ifdef \_\_SIZE\_TYPE\_\_

# define YYSIZE\_T \_\_SIZE\_TYPE\_\_

# elif defined size\_t

# define YYSIZE\_T size\_t

# elif defined \_\_STDC\_VERSION\_\_ && 199901 <= \_\_STDC\_VERSION\_\_

# include <stddef.h> /\* INFRINGES ON USER NAME SPACE \*/

# define YYSIZE\_T size\_t

# else

# define YYSIZE\_T unsigned

# endif

#endif

#define YYSIZE\_MAXIMUM \

YY\_CAST (YYPTRDIFF\_T, \

(YYPTRDIFF\_MAXIMUM < YY\_CAST (YYSIZE\_T, -1) \

? YYPTRDIFF\_MAXIMUM \

: YY\_CAST (YYSIZE\_T, -1)))

#define YYSIZEOF(X) YY\_CAST (YYPTRDIFF\_T, sizeof (X))

/\* Stored state numbers (used for stacks). \*/

typedef yytype\_int8 yy\_state\_t;

/\* State numbers in computations. \*/

typedef int yy\_state\_fast\_t;

#ifndef YY\_

# if defined YYENABLE\_NLS && YYENABLE\_NLS

# if ENABLE\_NLS

# include <libintl.h> /\* INFRINGES ON USER NAME SPACE \*/

# define YY\_(Msgid) dgettext ("bison-runtime", Msgid)

# endif

# endif

# ifndef YY\_

# define YY\_(Msgid) Msgid

# endif

#endif

#ifndef YY\_ATTRIBUTE\_PURE

# if defined \_\_GNUC\_\_ && 2 < \_\_GNUC\_\_ + (96 <= \_\_GNUC\_MINOR\_\_)

# define YY\_ATTRIBUTE\_PURE \_\_attribute\_\_ ((\_\_pure\_\_))

# else

# define YY\_ATTRIBUTE\_PURE

# endif

#endif

#ifndef YY\_ATTRIBUTE\_UNUSED

# if defined \_\_GNUC\_\_ && 2 < \_\_GNUC\_\_ + (7 <= \_\_GNUC\_MINOR\_\_)

# define YY\_ATTRIBUTE\_UNUSED \_\_attribute\_\_ ((\_\_unused\_\_))

# else

# define YY\_ATTRIBUTE\_UNUSED

# endif

#endif

/\* Suppress unused-variable warnings by "using" E. \*/

#if ! defined lint || defined \_\_GNUC\_\_

# define YYUSE(E) ((void) (E))

#else

# define YYUSE(E) /\* empty \*/

#endif

#if defined \_\_GNUC\_\_ && ! defined \_\_ICC && 407 <= \_\_GNUC\_\_ \* 100 + \_\_GNUC\_MINOR\_\_

/\* Suppress an incorrect diagnostic about yylval being uninitialized. \*/

# define YY\_IGNORE\_MAYBE\_UNINITIALIZED\_BEGIN \

\_Pragma ("GCC diagnostic push") \

\_Pragma ("GCC diagnostic ignored \"-Wuninitialized\"") \

\_Pragma ("GCC diagnostic ignored \"-Wmaybe-uninitialized\"")

# define YY\_IGNORE\_MAYBE\_UNINITIALIZED\_END \

\_Pragma ("GCC diagnostic pop")

#else

# define YY\_INITIAL\_VALUE(Value) Value

#endif

#ifndef YY\_IGNORE\_MAYBE\_UNINITIALIZED\_BEGIN

# define YY\_IGNORE\_MAYBE\_UNINITIALIZED\_BEGIN

# define YY\_IGNORE\_MAYBE\_UNINITIALIZED\_END

#endif

#ifndef YY\_INITIAL\_VALUE

# define YY\_INITIAL\_VALUE(Value) /\* Nothing. \*/

#endif

#if defined \_\_cplusplus && defined \_\_GNUC\_\_ && ! defined \_\_ICC && 6 <= \_\_GNUC\_\_

# define YY\_IGNORE\_USELESS\_CAST\_BEGIN \

\_Pragma ("GCC diagnostic push") \

\_Pragma ("GCC diagnostic ignored \"-Wuseless-cast\"")

# define YY\_IGNORE\_USELESS\_CAST\_END \

\_Pragma ("GCC diagnostic pop")

#endif

#ifndef YY\_IGNORE\_USELESS\_CAST\_BEGIN

# define YY\_IGNORE\_USELESS\_CAST\_BEGIN

# define YY\_IGNORE\_USELESS\_CAST\_END

#endif

#define YY\_ASSERT(E) ((void) (0 && (E)))

#if ! defined yyoverflow || YYERROR\_VERBOSE

/\* The parser invokes alloca or malloc; define the necessary symbols. \*/

# ifdef YYSTACK\_USE\_ALLOCA

# if YYSTACK\_USE\_ALLOCA

# ifdef \_\_GNUC\_\_

# define YYSTACK\_ALLOC \_\_builtin\_alloca

# elif defined \_\_BUILTIN\_VA\_ARG\_INCR

# include <alloca.h> /\* INFRINGES ON USER NAME SPACE \*/

# elif defined \_AIX

# define YYSTACK\_ALLOC \_\_alloca

# elif defined \_MSC\_VER

# include <malloc.h> /\* INFRINGES ON USER NAME SPACE \*/

# define alloca \_alloca

# else

# define YYSTACK\_ALLOC alloca

# if ! defined \_ALLOCA\_H && ! defined EXIT\_SUCCESS

# include <stdlib.h> /\* INFRINGES ON USER NAME SPACE \*/

/\* Use EXIT\_SUCCESS as a witness for stdlib.h. \*/

# ifndef EXIT\_SUCCESS

# define EXIT\_SUCCESS 0

# endif

# endif

# endif

# endif

# endif

# ifdef YYSTACK\_ALLOC

/\* Pacify GCC's 'empty if-body' warning. \*/

# define YYSTACK\_FREE(Ptr) do { /\* empty \*/; } while (0)

# ifndef YYSTACK\_ALLOC\_MAXIMUM

/\* The OS might guarantee only one guard page at the bottom of the stack,

and a page size can be as small as 4096 bytes. So we cannot safely

invoke alloca (N) if N exceeds 4096. Use a slightly smaller number

to allow for a few compiler-allocated temporary stack slots. \*/

# define YYSTACK\_ALLOC\_MAXIMUM 4032 /\* reasonable circa 2006 \*/

# endif

# else

# define YYSTACK\_ALLOC YYMALLOC

# define YYSTACK\_FREE YYFREE

# ifndef YYSTACK\_ALLOC\_MAXIMUM

# define YYSTACK\_ALLOC\_MAXIMUM YYSIZE\_MAXIMUM

# endif

# if (defined \_\_cplusplus && ! defined EXIT\_SUCCESS \

&& ! ((defined YYMALLOC || defined malloc) \

&& (defined YYFREE || defined free)))

# include <stdlib.h> /\* INFRINGES ON USER NAME SPACE \*/

# ifndef EXIT\_SUCCESS

# define EXIT\_SUCCESS 0

# endif

# endif

# ifndef YYMALLOC

# define YYMALLOC malloc

# if ! defined malloc && ! defined EXIT\_SUCCESS

void \*malloc (YYSIZE\_T); /\* INFRINGES ON USER NAME SPACE \*/

# endif

# endif

# ifndef YYFREE

# define YYFREE free

# if ! defined free && ! defined EXIT\_SUCCESS

void free (void \*); /\* INFRINGES ON USER NAME SPACE \*/

# endif

# endif

# endif

#endif /\* ! defined yyoverflow || YYERROR\_VERBOSE \*/

#if (! defined yyoverflow \

&& (! defined \_\_cplusplus \

|| (defined YYSTYPE\_IS\_TRIVIAL && YYSTYPE\_IS\_TRIVIAL)))

/\* A type that is properly aligned for any stack member. \*/

union **yyalloc**

{

yy\_state\_t yyss\_alloc;

YYSTYPE yyvs\_alloc;

};

/\* The size of the maximum gap between one aligned stack and the next. \*/

# define YYSTACK\_GAP\_MAXIMUM (YYSIZEOF (union yyalloc) - 1)

/\* The size of an array large to enough to hold all stacks, each with

N elements. \*/

# define YYSTACK\_BYTES(N) \

((N) \* (YYSIZEOF (yy\_state\_t) + YYSIZEOF (YYSTYPE)) \

+ YYSTACK\_GAP\_MAXIMUM)

# define YYCOPY\_NEEDED 1

/\* Relocate STACK from its old location to the new one. The

local variables YYSIZE and YYSTACKSIZE give the old and new number of

elements in the stack, and YYPTR gives the new location of the

stack. Advance YYPTR to a properly aligned location for the next

stack. \*/

# define YYSTACK\_RELOCATE(Stack\_alloc, Stack) \

do \

{ \

YYPTRDIFF\_T yynewbytes; \

YYCOPY (&yyptr->Stack\_alloc, Stack, yysize); \

Stack = &yyptr->Stack\_alloc; \

yynewbytes = yystacksize \* YYSIZEOF (\*Stack) + YYSTACK\_GAP\_MAXIMUM; \

yyptr += yynewbytes / YYSIZEOF (\*yyptr); \

} \

while (0)

#endif

#if defined YYCOPY\_NEEDED && YYCOPY\_NEEDED

/\* Copy COUNT objects from SRC to DST. The source and destination do

not overlap. \*/

# ifndef YYCOPY

# if defined \_\_GNUC\_\_ && 1 < \_\_GNUC\_\_

# define YYCOPY(Dst, Src, Count) \

\_\_builtin\_memcpy (Dst, Src, YY\_CAST (YYSIZE\_T, (Count)) \* sizeof (\*(Src)))

# else

# define YYCOPY(Dst, Src, Count) \

do \

{ \

YYPTRDIFF\_T yyi; \

for (yyi = 0; yyi < (Count); yyi++) \

(Dst)[yyi] = (Src)[yyi]; \

} \

while (0)

# endif

# endif

#endif /\* !YYCOPY\_NEEDED \*/

/\* YYFINAL -- State number of the termination state. \*/

#define YYFINAL 9

/\* YYLAST -- Last index in YYTABLE. \*/

#define YYLAST 29

/\* YYNTOKENS -- Number of terminals. \*/

#define YYNTOKENS 12

/\* YYNNTS -- Number of nonterminals. \*/

#define YYNNTS 9

/\* YYNRULES -- Number of rules. \*/

#define YYNRULES 24

/\* YYNSTATES -- Number of states. \*/

#define YYNSTATES 30

#define YYUNDEFTOK 2

#define YYMAXUTOK 266

/\* YYTRANSLATE(TOKEN-NUM) -- Symbol number corresponding to TOKEN-NUM

as returned by yylex, with out-of-bounds checking. \*/

#define YYTRANSLATE(YYX) \

(0 <= (YYX) && (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)

/\* YYTRANSLATE[TOKEN-NUM] -- Symbol number corresponding to TOKEN-NUM

as returned by yylex. \*/

static const yytype\_int8 yytranslate[] =

{

0, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 1, 2, 3, 4,

5, 6, 7, 8, 9, 10, 11

};

#if YYDEBUG

/\* YYRLINE[YYN] -- Source line where rule number YYN was defined. \*/

static const yytype\_int8 yyrline[] =

{

0, 18, 18, 18, 18, 18, 20, 20, 22, 22,

24, 24, 24, 26, 26, 28, 28, 28, 30, 30,

30, 32, 32, 32, 32

};

#endif

#if YYDEBUG || YYERROR\_VERBOSE || 0

/\* YYTNAME[SYMBOL-NUM] -- String name of the symbol SYMBOL-NUM.

First, the terminals, then, starting at YYNTOKENS, nonterminals. \*/

static const char \*const yytname[] =

{

"$end", "error", "$undefined", "FINSENT", "READ", "DIGIT", "SYMBOL",

"MULTI", "A\_PARENT", "C\_PARENT", "NEWLINE", "COMMA", "$accept", "PROG",

"EXPR", "TERM", "VALUE", "OPER", "FMT", "INT", "ID", YY\_NULLPTR

};

#endif

# ifdef YYPRINT

/\* YYTOKNUM[NUM] -- (External) token number corresponding to the

(internal) symbol number NUM (which must be that of a token). \*/

static const yytype\_int16 yytoknum[] =

{

0, 256, 257, 258, 259, 260, 261, 262, 263, 264,

265, 266

};

# endif

#define YYPACT\_NINF (-5)

#define yypact\_value\_is\_default(Yyn) \

((Yyn) == YYPACT\_NINF)

#define YYTABLE\_NINF (-25)

#define yytable\_value\_is\_error(Yyn) \

0

/\* YYPACT[STATE-NUM] -- Index in YYTABLE of the portion describing

STATE-NUM. \*/

static const yytype\_int8 yypact[] =

{

0, -5, 6, 2, -5, 3, -4, 5, 15, -5,

-5, 9, -5, 13, -5, 14, 17, 10, 15, -5,

-5, -5, 4, -5, 18, -5, 14, -5, -5, -5

};

/\* YYDEFACT[STATE-NUM] -- Default reduction number in state STATE-NUM.

Performed when YYTABLE does not specify something else to do. Zero

means the default is an error. \*/

static const yytype\_int8 yydefact[] =

{

0, 2, 0, 0, 3, 14, 0, 0, 0, 1,

4, 0, 6, 12, 21, 8, 10, 0, 0, 22,

23, 17, 0, 15, 0, 16, 11, 20, 19, 13

};

/\* YYPGOTO[NTERM-NUM]. \*/

static const yytype\_int8 yypgoto[] =

{

-5, -5, -5, -5, 8, -5, -5, 7, -5

};

/\* YYDEFGOTO[NTERM-NUM]. \*/

static const yytype\_int8 yydefgoto[] =

{

-1, 3, 4, 7, 15, 8, 24, 25, 16

};

/\* YYTABLE[YYPACT[STATE-NUM]] -- What to do in state STATE-NUM. If

positive, shift that token. If negative, reduce the rule whose

number is the opposite. If YYTABLE\_NINF, syntax error. \*/

static const yytype\_int8 yytable[] =

{

-5, 1, 9, 11, 2, 27, -9, 5, 12, 22,

-5, 21, 10, -18, 6, 22, 13, 23, -24, -24,

17, 14, 19, 20, 0, 18, 26, 29, 0, 28

};

static const yytype\_int8 yycheck[] =

{

0, 1, 0, 7, 4, 1, 3, 1, 3, 5,

10, 1, 10, 9, 8, 5, 1, 7, 5, 6,

11, 6, 5, 6, -1, 11, 18, 9, -1, 22

};

/\* YYSTOS[STATE-NUM] -- The (internal number of the) accessing

symbol of state STATE-NUM. \*/

static const yytype\_int8 yystos[] =

{

0, 1, 4, 13, 14, 1, 8, 15, 17, 0,

10, 7, 3, 1, 6, 16, 20, 11, 11, 5,

6, 1, 5, 7, 18, 19, 16, 1, 19, 9

};

/\* YYR1[YYN] -- Symbol number of symbol that rule YYN derives. \*/

static const yytype\_int8 yyr1[] =

{

0, 12, 13, 13, 13, 13, 14, 14, 15, 15,

16, 16, 16, 17, 17, 18, 18, 18, 19, 19,

19, 20, 20, 20, 20

};

/\* YYR2[YYN] -- Number of symbols on the right hand side of rule YYN. \*/

static const yytype\_int8 yyr2[] =

{

0, 2, 1, 1, 2, 0, 3, 1, 2, 1,

1, 3, 1, 5, 1, 1, 1, 1, 1, 2,

1, 1, 2, 2, 1

};

#define yyerrok (yyerrstatus = 0)

#define yyclearin (yychar = YYEMPTY)

#define YYEMPTY (-2)

#define YYEOF 0

#define YYACCEPT goto yyacceptlab

#define YYABORT goto yyabortlab

#define YYERROR goto yyerrorlab

#define YYRECOVERING() (!!yyerrstatus)

#define YYBACKUP(Token, Value) \

do \

if (yychar == YYEMPTY) \

{ \

yychar = (Token); \

yylval = (Value); \

YYPOPSTACK (yylen); \

yystate = \*yyssp; \

goto yybackup; \

} \

else \

{ \

yyerror (YY\_("syntax error: cannot back up")); \

YYERROR; \

} \

while (0)

/\* Error token number \*/

#define YYTERROR 1

#define YYERRCODE 256

/\* Enable debugging if requested. \*/

#if YYDEBUG

# ifndef YYFPRINTF

# include <stdio.h> /\* INFRINGES ON USER NAME SPACE \*/

# define YYFPRINTF fprintf

# endif

# define YYDPRINTF(Args) \

do { \

if (yydebug) \

YYFPRINTF Args; \

} while (0)

/\* This macro is provided for backward compatibility. \*/

#ifndef YY\_LOCATION\_PRINT

# define YY\_LOCATION\_PRINT(File, Loc) ((void) 0)

#endif

# define YY\_SYMBOL\_PRINT(Title, Type, Value, Location) \

do { \

if (yydebug) \

{ \

YYFPRINTF (stderr, "%s ", Title); \

yy\_symbol\_print (stderr, \

Type, Value); \

YYFPRINTF (stderr, "\n"); \

} \

} while (0)

/\*-----------------------------------.

| Print this symbol's value on YYO. |

`-----------------------------------\*/

static void

yy\_symbol\_value\_print (FILE \*yyo, int yytype, YYSTYPE const \* const yyvaluep)

{

FILE \*yyoutput = yyo;

YYUSE (yyoutput);

if (!yyvaluep)

return;

# ifdef YYPRINT

if (yytype < YYNTOKENS)

YYPRINT (yyo, yytoknum[yytype], \*yyvaluep);

# endif

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_BEGIN

YYUSE (yytype);

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_END

}

/\*---------------------------.

| Print this symbol on YYO. |

`---------------------------\*/

static void

yy\_symbol\_print (FILE \*yyo, int yytype, YYSTYPE const \* const yyvaluep)

{

YYFPRINTF (yyo, "%s %s (",

yytype < YYNTOKENS ? "token" : "nterm", yytname[yytype]);

yy\_symbol\_value\_print (yyo, yytype, yyvaluep);

YYFPRINTF (yyo, ")");

}

/\*------------------------------------------------------------------.

| yy\_stack\_print -- Print the state stack from its BOTTOM up to its |

| TOP (included). |

`------------------------------------------------------------------\*/

static void

yy\_stack\_print (yy\_state\_t \*yybottom, yy\_state\_t \*yytop)

{

YYFPRINTF (stderr, "Stack now");

for (; yybottom <= yytop; yybottom++)

{

int yybot = \*yybottom;

YYFPRINTF (stderr, " %d", yybot);

}

YYFPRINTF (stderr, "\n");

}

# define YY\_STACK\_PRINT(Bottom, Top) \

do { \

if (yydebug) \

yy\_stack\_print ((Bottom), (Top)); \

} while (0)

/\*------------------------------------------------.

| Report that the YYRULE is going to be reduced. |

`------------------------------------------------\*/

static void

yy\_reduce\_print (yy\_state\_t \*yyssp, YYSTYPE \*yyvsp, int yyrule)

{

int yylno = yyrline[yyrule];

int yynrhs = yyr2[yyrule];

int yyi;

YYFPRINTF (stderr, "Reducing stack by rule %d (line %d):\n",

yyrule - 1, yylno);

/\* The symbols being reduced. \*/

for (yyi = 0; yyi < yynrhs; yyi++)

{

YYFPRINTF (stderr, " $%d = ", yyi + 1);

yy\_symbol\_print (stderr,

yystos[+yyssp[yyi + 1 - yynrhs]],

&yyvsp[(yyi + 1) - (yynrhs)]

);

YYFPRINTF (stderr, "\n");

}

}

# define YY\_REDUCE\_PRINT(Rule) \

do { \

if (yydebug) \

yy\_reduce\_print (yyssp, yyvsp, Rule); \

} while (0)

/\* Nonzero means print parse trace. It is left uninitialized so that

multiple parsers can coexist. \*/

int yydebug;

#else /\* !YYDEBUG \*/

# define YYDPRINTF(Args)

# define YY\_SYMBOL\_PRINT(Title, Type, Value, Location)

# define YY\_STACK\_PRINT(Bottom, Top)

# define YY\_REDUCE\_PRINT(Rule)

#endif /\* !YYDEBUG \*/

/\* YYINITDEPTH -- initial size of the parser's stacks. \*/

#ifndef YYINITDEPTH

# define YYINITDEPTH 200

#endif

/\* YYMAXDEPTH -- maximum size the stacks can grow to (effective only

if the built-in stack extension method is used).

Do not make this value too large; the results are undefined if

YYSTACK\_ALLOC\_MAXIMUM < YYSTACK\_BYTES (YYMAXDEPTH)

evaluated with infinite-precision integer arithmetic. \*/

#ifndef YYMAXDEPTH

# define YYMAXDEPTH 10000

#endif

#if YYERROR\_VERBOSE

# ifndef yystrlen

# if defined \_\_GLIBC\_\_ && defined \_STRING\_H

# define yystrlen(S) (YY\_CAST (YYPTRDIFF\_T, strlen (S)))

# else

/\* Return the length of YYSTR. \*/

static YYPTRDIFF\_T

yystrlen (const char \*yystr)

{

YYPTRDIFF\_T yylen;

for (yylen = 0; yystr[yylen]; yylen++)

continue;

return yylen;

}

# endif

# endif

# ifndef yystpcpy

# if defined \_\_GLIBC\_\_ && defined \_STRING\_H && defined \_GNU\_SOURCE

# define yystpcpy stpcpy

# else

/\* Copy YYSRC to YYDEST, returning the address of the terminating '\0' in

YYDEST. \*/

static char \*

yystpcpy (char \*yydest, const char \*yysrc)

{

char \*yyd = yydest;

const char \*yys = yysrc;

while ((\*yyd++ = \*yys++) != '\0')

continue;

return yyd - 1;

}

# endif

# endif

# ifndef yytnamerr

/\* Copy to YYRES the contents of YYSTR after stripping away unnecessary

quotes and backslashes, so that it's suitable for yyerror. The

heuristic is that double-quoting is unnecessary unless the string

contains an apostrophe, a comma, or backslash (other than

backslash-backslash). YYSTR is taken from yytname. If YYRES is

null, do not copy; instead, return the length of what the result

would have been. \*/

static YYPTRDIFF\_T

yytnamerr (char \*yyres, const char \*yystr)

{

if (\*yystr == '"')

{

YYPTRDIFF\_T yyn = 0;

char const \*yyp = yystr;

for (;;)

switch (\*++yyp)

{

case '\'':

case ',':

goto do\_not\_strip\_quotes;

case '\\':

if (\*++yyp != '\\')

goto do\_not\_strip\_quotes;

else

goto append;

append:

default:

if (yyres)

yyres[yyn] = \*yyp;

yyn++;

break;

case '"':

if (yyres)

yyres[yyn] = '\0';

return yyn;

}

do\_not\_strip\_quotes: ;

}

if (yyres)

return yystpcpy (yyres, yystr) - yyres;

else

return yystrlen (yystr);

}

# endif

/\* Copy into \*YYMSG, which is of size \*YYMSG\_ALLOC, an error message

about the unexpected token YYTOKEN for the state stack whose top is

YYSSP.

Return 0 if \*YYMSG was successfully written. Return 1 if \*YYMSG is

not large enough to hold the message. In that case, also set

\*YYMSG\_ALLOC to the required number of bytes. Return 2 if the

required number of bytes is too large to store. \*/

static int

yysyntax\_error (YYPTRDIFF\_T \*yymsg\_alloc, char \*\*yymsg,

yy\_state\_t \*yyssp, int yytoken)

{

enum { YYERROR\_VERBOSE\_ARGS\_MAXIMUM = 5 };

/\* Internationalized format string. \*/

const char \*yyformat = YY\_NULLPTR;

/\* Arguments of yyformat: reported tokens (one for the "unexpected",

one per "expected"). \*/

char const \*yyarg[YYERROR\_VERBOSE\_ARGS\_MAXIMUM];

/\* Actual size of YYARG. \*/

int yycount = 0;

/\* Cumulated lengths of YYARG. \*/

YYPTRDIFF\_T yysize = 0;

/\* There are many possibilities here to consider:

- If this state is a consistent state with a default action, then

the only way this function was invoked is if the default action

is an error action. In that case, don't check for expected

tokens because there are none.

- The only way there can be no lookahead present (in yychar) is if

this state is a consistent state with a default action. Thus,

detecting the absence of a lookahead is sufficient to determine

that there is no unexpected or expected token to report. In that

case, just report a simple "syntax error".

- Don't assume there isn't a lookahead just because this state is a

consistent state with a default action. There might have been a

previous inconsistent state, consistent state with a non-default

action, or user semantic action that manipulated yychar.

- Of course, the expected token list depends on states to have

correct lookahead information, and it depends on the parser not

to perform extra reductions after fetching a lookahead from the

scanner and before detecting a syntax error. Thus, state merging

(from LALR or IELR) and default reductions corrupt the expected

token list. However, the list is correct for canonical LR with

one exception: it will still contain any token that will not be

accepted due to an error action in a later state.

\*/

if (yytoken != YYEMPTY)

{

int yyn = yypact[+\*yyssp];

YYPTRDIFF\_T yysize0 = yytnamerr (YY\_NULLPTR, yytname[yytoken]);

yysize = yysize0;

yyarg[yycount++] = yytname[yytoken];

if (!yypact\_value\_is\_default (yyn))

{

/\* Start YYX at -YYN if negative to avoid negative indexes in

YYCHECK. In other words, skip the first -YYN actions for

this state because they are default actions. \*/

int yyxbegin = yyn < 0 ? -yyn : 0;

/\* Stay within bounds of both yycheck and yytname. \*/

int yychecklim = YYLAST - yyn + 1;

int yyxend = yychecklim < YYNTOKENS ? yychecklim : YYNTOKENS;

int yyx;

for (yyx = yyxbegin; yyx < yyxend; ++yyx)

if (yycheck[yyx + yyn] == yyx && yyx != YYTERROR

&& !yytable\_value\_is\_error (yytable[yyx + yyn]))

{

if (yycount == YYERROR\_VERBOSE\_ARGS\_MAXIMUM)

{

yycount = 1;

yysize = yysize0;

break;

}

yyarg[yycount++] = yytname[yyx];

{

YYPTRDIFF\_T yysize1

= yysize + yytnamerr (YY\_NULLPTR, yytname[yyx]);

if (yysize <= yysize1 && yysize1 <= YYSTACK\_ALLOC\_MAXIMUM)

yysize = yysize1;

else

return 2;

}

}

}

}

switch (yycount)

{

# define YYCASE\_(N, S) \

case N: \

yyformat = S; \

break

default: /\* Avoid compiler warnings. \*/

YYCASE\_(0, YY\_("syntax error"));

YYCASE\_(1, YY\_("syntax error, unexpected %s"));

YYCASE\_(2, YY\_("syntax error, unexpected %s, expecting %s"));

YYCASE\_(3, YY\_("syntax error, unexpected %s, expecting %s or %s"));

YYCASE\_(4, YY\_("syntax error, unexpected %s, expecting %s or %s or %s"));

YYCASE\_(5, YY\_("syntax error, unexpected %s, expecting %s or %s or %s or %s"));

# undef YYCASE\_

}

{

/\* Don't count the "%s"s in the final size, but reserve room for

the terminator. \*/

YYPTRDIFF\_T yysize1 = yysize + (yystrlen (yyformat) - 2 \* yycount) + 1;

if (yysize <= yysize1 && yysize1 <= YYSTACK\_ALLOC\_MAXIMUM)

yysize = yysize1;

else

return 2;

}

if (\*yymsg\_alloc < yysize)

{

\*yymsg\_alloc = 2 \* yysize;

if (! (yysize <= \*yymsg\_alloc

&& \*yymsg\_alloc <= YYSTACK\_ALLOC\_MAXIMUM))

\*yymsg\_alloc = YYSTACK\_ALLOC\_MAXIMUM;

return 1;

}

/\* Avoid sprintf, as that infringes on the user's name space.

Don't have undefined behavior even if the translation

produced a string with the wrong number of "%s"s. \*/

{

char \*yyp = \*yymsg;

int yyi = 0;

while ((\*yyp = \*yyformat) != '\0')

if (\*yyp == '%' && yyformat[1] == 's' && yyi < yycount)

{

yyp += yytnamerr (yyp, yyarg[yyi++]);

yyformat += 2;

}

else

{

++yyp;

++yyformat;

}

}

return 0;

}

#endif /\* YYERROR\_VERBOSE \*/

/\*-----------------------------------------------.

| Release the memory associated to this symbol. |

`-----------------------------------------------\*/

static void

**yydestruct** (const char \*yymsg, int yytype, YYSTYPE \*yyvaluep)

{

YYUSE (yyvaluep);

if (!yymsg)

yymsg = "Deleting";

YY\_SYMBOL\_PRINT (yymsg, yytype, yyvaluep, yylocationp);

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_BEGIN

YYUSE (yytype);

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_END

}

/\* The lookahead symbol. \*/

int yychar;

/\* The semantic value of the lookahead symbol. \*/

YYSTYPE yylval;

/\* Number of syntax errors so far. \*/

int yynerrs;

/\*----------.

| yyparse. |

`----------\*/

int

**yyparse** (void)

{

yy\_state\_fast\_t yystate;

/\* Number of tokens to shift before error messages enabled. \*/

int yyerrstatus;

/\* The stacks and their tools:

'yyss': related to states.

'yyvs': related to semantic values.

Refer to the stacks through separate pointers, to allow yyoverflow

to reallocate them elsewhere. \*/

/\* The state stack. \*/

yy\_state\_t yyssa[YYINITDEPTH];

yy\_state\_t \*yyss;

yy\_state\_t \*yyssp;

/\* The semantic value stack. \*/

YYSTYPE yyvsa[YYINITDEPTH];

YYSTYPE \*yyvs;

YYSTYPE \*yyvsp;

YYPTRDIFF\_T yystacksize;

int yyn;

int yyresult;

/\* Lookahead token as an internal (translated) token number. \*/

int yytoken = 0;

/\* The variables used to return semantic value and location from the

action routines. \*/

YYSTYPE yyval;

#if YYERROR\_VERBOSE

/\* Buffer for error messages, and its allocated size. \*/

char yymsgbuf[128];

char \*yymsg = yymsgbuf;

YYPTRDIFF\_T yymsg\_alloc = sizeof yymsgbuf;

#endif

#define YYPOPSTACK(N) (yyvsp -= (N), yyssp -= (N))

/\* The number of symbols on the RHS of the reduced rule.

Keep to zero when no symbol should be popped. \*/

int yylen = 0;

yyssp = yyss = yyssa;

yyvsp = yyvs = yyvsa;

yystacksize = YYINITDEPTH;

YYDPRINTF ((stderr, "Starting parse\n"));

yystate = 0;

yyerrstatus = 0;

yynerrs = 0;

yychar = YYEMPTY; /\* Cause a token to be read. \*/

goto yysetstate;

/\*------------------------------------------------------------.

| yynewstate -- push a new state, which is found in yystate. |

`------------------------------------------------------------\*/

yynewstate:

/\* In all cases, when you get here, the value and location stacks

have just been pushed. So pushing a state here evens the stacks. \*/

yyssp++;

/\*--------------------------------------------------------------------.

| yysetstate -- set current state (the top of the stack) to yystate. |

`--------------------------------------------------------------------\*/

yysetstate:

YYDPRINTF ((stderr, "Entering state %d\n", yystate));

YY\_ASSERT (0 <= yystate && yystate < YYNSTATES);

YY\_IGNORE\_USELESS\_CAST\_BEGIN

\*yyssp = YY\_CAST (yy\_state\_t, yystate);

YY\_IGNORE\_USELESS\_CAST\_END

if (yyss + yystacksize - 1 <= yyssp)

#if !defined yyoverflow && !defined YYSTACK\_RELOCATE

goto yyexhaustedlab;

#else

{

/\* Get the current used size of the three stacks, in elements. \*/

YYPTRDIFF\_T yysize = yyssp - yyss + 1;

# if defined yyoverflow

{

/\* Give user a chance to reallocate the stack. Use copies of

these so that the &'s don't force the real ones into

memory. \*/

yy\_state\_t \*yyss1 = yyss;

YYSTYPE \*yyvs1 = yyvs;

/\* Each stack pointer address is followed by the size of the

data in use in that stack, in bytes. This used to be a

conditional around just the two extra args, but that might

be undefined if yyoverflow is a macro. \*/

yyoverflow (YY\_("memory exhausted"),

&yyss1, yysize \* YYSIZEOF (\*yyssp),

&yyvs1, yysize \* YYSIZEOF (\*yyvsp),

&yystacksize);

yyss = yyss1;

yyvs = yyvs1;

}

# else /\* defined YYSTACK\_RELOCATE \*/

/\* Extend the stack our own way. \*/

if (YYMAXDEPTH <= yystacksize)

goto yyexhaustedlab;

yystacksize \*= 2;

if (YYMAXDEPTH < yystacksize)

yystacksize = YYMAXDEPTH;

{

yy\_state\_t \*yyss1 = yyss;

union yyalloc \*yyptr =

YY\_CAST (union yyalloc \*,

YYSTACK\_ALLOC (YY\_CAST (YYSIZE\_T, YYSTACK\_BYTES (yystacksize))));

if (! yyptr)

goto yyexhaustedlab;

YYSTACK\_RELOCATE (yyss\_alloc, yyss);

YYSTACK\_RELOCATE (yyvs\_alloc, yyvs);

# undef YYSTACK\_RELOCATE

if (yyss1 != yyssa)

YYSTACK\_FREE (yyss1);

}

# endif

yyssp = yyss + yysize - 1;

yyvsp = yyvs + yysize - 1;

YY\_IGNORE\_USELESS\_CAST\_BEGIN

YYDPRINTF ((stderr, "Stack size increased to %ld\n",

YY\_CAST (long, yystacksize)));

YY\_IGNORE\_USELESS\_CAST\_END

if (yyss + yystacksize - 1 <= yyssp)

YYABORT;

}

#endif /\* !defined yyoverflow && !defined YYSTACK\_RELOCATE \*/

if (yystate == YYFINAL)

YYACCEPT;

goto yybackup;

/\*-----------.

| yybackup. |

`-----------\*/

yybackup:

/\* Do appropriate processing given the current state. Read a

lookahead token if we need one and don't already have one. \*/

/\* First try to decide what to do without reference to lookahead token. \*/

yyn = yypact[yystate];

if (yypact\_value\_is\_default (yyn))

goto yydefault;

/\* Not known => get a lookahead token if don't already have one. \*/

/\* YYCHAR is either YYEMPTY or YYEOF or a valid lookahead symbol. \*/

if (yychar == YYEMPTY)

{

YYDPRINTF ((stderr, "Reading a token: "));

yychar = yylex ();

}

if (yychar <= YYEOF)

{

yychar = yytoken = YYEOF;

YYDPRINTF ((stderr, "Now at end of input.\n"));

}

else

{

yytoken = YYTRANSLATE (yychar);

YY\_SYMBOL\_PRINT ("Next token is", yytoken, &yylval, &yylloc);

}

/\* If the proper action on seeing token YYTOKEN is to reduce or to

detect an error, take that action. \*/

yyn += yytoken;

if (yyn < 0 || YYLAST < yyn || yycheck[yyn] != yytoken)

goto yydefault;

yyn = yytable[yyn];

if (yyn <= 0)

{

if (yytable\_value\_is\_error (yyn))

goto yyerrlab;

yyn = -yyn;

goto yyreduce;

}

/\* Count tokens shifted since error; after three, turn off error

status. \*/

if (yyerrstatus)

yyerrstatus--;

/\* Shift the lookahead token. \*/

YY\_SYMBOL\_PRINT ("Shifting", yytoken, &yylval, &yylloc);

yystate = yyn;

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_BEGIN

\*++yyvsp = yylval;

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_END

/\* Discard the shifted token. \*/

yychar = YYEMPTY;

goto yynewstate;

/\*-----------------------------------------------------------.

| yydefault -- do the default action for the current state. |

`-----------------------------------------------------------\*/

yydefault:

yyn = yydefact[yystate];

if (yyn == 0)

goto yyerrlab;

goto yyreduce;

/\*-----------------------------.

| yyreduce -- do a reduction. |

`-----------------------------\*/

yyreduce:

/\* yyn is the number of a rule to reduce with. \*/

yylen = yyr2[yyn];

/\* If YYLEN is nonzero, implement the default value of the action:

'$$ = $1'.

Otherwise, the following line sets YYVAL to garbage.

This behavior is undocumented and Bison

users should not rely upon it. Assigning to YYVAL

unconditionally makes the parser a bit smaller, and it avoids a

GCC warning that YYVAL may be used uninitialized. \*/

yyval = yyvsp[1-yylen];

YY\_REDUCE\_PRINT (yyn);

switch (yyn)

{

#line 1313 "readParser.c"

default: break;

}

/\* User semantic actions sometimes alter yychar, and that requires

that yytoken be updated with the new translation. We take the

approach of translating immediately before every use of yytoken.

One alternative is translating here after every semantic action,

but that translation would be missed if the semantic action invokes

YYABORT, YYACCEPT, or YYERROR immediately after altering yychar or

if it invokes YYBACKUP. In the case of YYABORT or YYACCEPT, an

incorrect destructor might then be invoked immediately. In the

case of YYERROR or YYBACKUP, subsequent parser actions might lead

to an incorrect destructor call or verbose syntax error message

before the lookahead is translated. \*/

YY\_SYMBOL\_PRINT ("-> $$ =", yyr1[yyn], &yyval, &yyloc);

YYPOPSTACK (yylen);

yylen = 0;

YY\_STACK\_PRINT (yyss, yyssp);

\*++yyvsp = yyval;

/\* Now 'shift' the result of the reduction. Determine what state

that goes to, based on the state we popped back to and the rule

number reduced by. \*/

{

const int yylhs = yyr1[yyn] - YYNTOKENS;

const int yyi = yypgoto[yylhs] + \*yyssp;

yystate = (0 <= yyi && yyi <= YYLAST && yycheck[yyi] == \*yyssp

? yytable[yyi]

: yydefgoto[yylhs]);

}

goto yynewstate;

/\*--------------------------------------.

| yyerrlab -- here on detecting error. |

`--------------------------------------\*/

yyerrlab:

/\* Make sure we have latest lookahead translation. See comments at

user semantic actions for why this is necessary. \*/

yytoken = yychar == YYEMPTY ? YYEMPTY : YYTRANSLATE (yychar);

/\* If not already recovering from an error, report this error. \*/

if (!yyerrstatus)

{

++yynerrs;

#if ! YYERROR\_VERBOSE

yyerror (YY\_("syntax error"));

#else

# define YYSYNTAX\_ERROR yysyntax\_error (&yymsg\_alloc, &yymsg, \

yyssp, yytoken)

{

char const \*yymsgp = YY\_("syntax error");

int yysyntax\_error\_status;

yysyntax\_error\_status = YYSYNTAX\_ERROR;

if (yysyntax\_error\_status == 0)

yymsgp = yymsg;

else if (yysyntax\_error\_status == 1)

{

if (yymsg != yymsgbuf)

YYSTACK\_FREE (yymsg);

yymsg = YY\_CAST (char \*, YYSTACK\_ALLOC (YY\_CAST (YYSIZE\_T, yymsg\_alloc)));

if (!yymsg)

{

yymsg = yymsgbuf;

yymsg\_alloc = sizeof yymsgbuf;

yysyntax\_error\_status = 2;

}

else

{

yysyntax\_error\_status = YYSYNTAX\_ERROR;

yymsgp = yymsg;

}

}

yyerror (yymsgp);

if (yysyntax\_error\_status == 2)

goto yyexhaustedlab;

}

# undef YYSYNTAX\_ERROR

#endif

}

if (yyerrstatus == 3)

{

/\* If just tried and failed to reuse lookahead token after an

error, discard it. \*/

if (yychar <= YYEOF)

{

/\* Return failure if at end of input. \*/

if (yychar == YYEOF)

YYABORT;

}

else

{

yydestruct ("Error: discarding",

yytoken, &yylval);

yychar = YYEMPTY;

}

}

/\* Else will try to reuse lookahead token after shifting the error

token. \*/

goto yyerrlab1;

/\*---------------------------------------------------.

| yyerrorlab -- error raised explicitly by YYERROR. |

`---------------------------------------------------\*/

yyerrorlab:

/\* Pacify compilers when the user code never invokes YYERROR and the

label yyerrorlab therefore never appears in user code. \*/

if (0)

YYERROR;

/\* Do not reclaim the symbols of the rule whose action triggered

this YYERROR. \*/

YYPOPSTACK (yylen);

yylen = 0;

YY\_STACK\_PRINT (yyss, yyssp);

yystate = \*yyssp;

goto yyerrlab1;

/\*-------------------------------------------------------------.

| yyerrlab1 -- common code for both syntax error and YYERROR. |

`-------------------------------------------------------------\*/

yyerrlab1:

yyerrstatus = 3; /\* Each real token shifted decrements this. \*/

for (;;)

{

yyn = yypact[yystate];

if (!yypact\_value\_is\_default (yyn))

{

yyn += YYTERROR;

if (0 <= yyn && yyn <= YYLAST && yycheck[yyn] == YYTERROR)

{

yyn = yytable[yyn];

if (0 < yyn)

break;

}

}

/\* Pop the current state because it cannot handle the error token. \*/

if (yyssp == yyss)

YYABORT;

yydestruct ("Error: popping",

yystos[yystate], yyvsp);

YYPOPSTACK (1);

yystate = \*yyssp;

YY\_STACK\_PRINT (yyss, yyssp);

}

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_BEGIN

\*++yyvsp = yylval;

YY\_IGNORE\_MAYBE\_UNINITIALIZED\_END

/\* Shift the error token. \*/

YY\_SYMBOL\_PRINT ("Shifting", yystos[yyn], yyvsp, yylsp);

yystate = yyn;

goto yynewstate;

/\*-------------------------------------.

| yyacceptlab -- YYACCEPT comes here. |

`-------------------------------------\*/

yyacceptlab:

yyresult = 0;

goto yyreturn;

/\*-----------------------------------.

| yyabortlab -- YYABORT comes here. |

`-----------------------------------\*/

yyabortlab:

yyresult = 1;

goto yyreturn;

#if !defined yyoverflow || YYERROR\_VERBOSE

/\*-------------------------------------------------.

| yyexhaustedlab -- memory exhaustion comes here. |

`-------------------------------------------------\*/

yyexhaustedlab:

yyerror (YY\_("memory exhausted"));

yyresult = 2;

/\* Fall through. \*/

#endif

/\*-----------------------------------------------------.

| yyreturn -- parsing is finished, return the result. |

`-----------------------------------------------------\*/

yyreturn:

if (yychar != YYEMPTY)

{

/\* Make sure we have latest lookahead translation. See comments at

user semantic actions for why this is necessary. \*/

yytoken = YYTRANSLATE (yychar);

yydestruct ("Cleanup: discarding lookahead",

yytoken, &yylval);

}

/\* Do not reclaim the symbols of the rule whose action triggered

this YYABORT or YYACCEPT. \*/

YYPOPSTACK (yylen);

YY\_STACK\_PRINT (yyss, yyssp);

while (yyssp != yyss)

{

yydestruct ("Cleanup: popping",

yystos[+\*yyssp], yyvsp);

YYPOPSTACK (1);

}

#ifndef yyoverflow

if (yyss != yyssa)

YYSTACK\_FREE (yyss);

#endif

#if YYERROR\_VERBOSE

if (yymsg != yymsgbuf)

YYSTACK\_FREE (yymsg);

#endif

return yyresult;

}

#line 34 "GRAM.y"

int countErr = 0;

int **main** (int argc, char \*\*argv)

{

extern FILE \*yyin;

yyin = fopen(argv[1], "rt");

if(yyin == NULL)

{

printf("No se puede leer el archivo seleccionado\n");

return 0;

}

yyparse();

if(countErr != 0)

printf("\nCount errors: %d\n",countErr);

else

printf("NO ERROR\n");

fclose(yyin);

}

yyerror(char \*s) {

countErr++;

extern char \*yytext;

extern int yylineno;

fprintf(stderr,"%d) ERROR (line %d):before '%s'-%s \n", countErr, yylineno, yytext, s);

}