Министерство образования и науки Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

**Институт компьютерных наук и кибербезопасности**   
  
  
  
**КУРСОВАЯ РАБОТА**  
  
**«Разработка парсера языка Ruby»**  
**по дисциплине «Формальные грамматики и теория компиляторов»**

Выполнили Подоров А.А.

студенты гр. 5151003/00801 Ясинский А.Н.

Руководитель

ст. преподаватель Семьянов П.В.

Санкт-Петербург

2023

**ОГЛАВЛЕНИЕ**

[1 ЦЕЛЬ РАБОТЫ 2](#_Toc1833106241)

[2 ЗАДАЧИ 3](#_Toc1184763687)

[3 ХОД РАБОТЫ 4](#_Toc2068249550)

[3.1 Теоретические сведения 5](#_Toc1126321356)

[3.2 Лексический анализатор языка 6](#_Toc87440606)

[3.3 Синтаксический анализатор языка 8](#_Toc2039116862)

[3.4 Возникшие конфликты и их исправление 14](#_Toc902314393)

[3.5 Работоспособность парсера 16](#_Toc272393294)

[5 ВЫВОД 19](#_Toc971270458)

[ПРИЛОЖЕНИЕ 1 21](#_Toc680514427)

[ПРИЛОЖЕНИЕ 2 27](#_Toc1909549639)

# ЦЕЛЬ РАБОТЫ

Получить навыки работы со средствами разработки парсеров языков программирования, а также построения грамматик, которые позволят распознать принадлежность входного кода к существующему языку программирования.

# ЗАДАЧИ

Разработать парсер языка Ruby используя возможности средств лексического и синтаксического анализа yacc и flex. Составить грамматику, позволяющую распознавать язык программирования Ruby.

# ХОД РАБОТЫ

## Теоретические сведения

**Ruby** — [динамический, высокоуровневый](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Язык по особенностям [синтаксиса](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81) близок к языку Perl, по объектно-ориентированному подходу — к [Smal](https://ru.wikipedia.org/wiki/Smalltalk)ltalk. Также некоторые черты языка взяты из [Python](https://ru.wikipedia.org/wiki/Python).

**Характерные особенности языка Ruby:**

* Всё в Ruby – объекты;
* Ruby очень гибкий язык, так как он позволяет его пользователям свободно менять его части;

**Например:** сложение выполняется операцией плюс (+). Но, если вы хотите использовать для этого более читаемое слово plus – вы можете добавить такой метод (plus).

* Ruby не нуждается в объявлении переменных. В нем используются простые соглашения по именованию, для того чтобы разделить области видимости переменных

**Например:** var— может быть локальной переменной, @var— переменная объекта, $var— глобальная переменная.

Язык Ruby похож на язык Python тем, что имеется строгая динамическая типизация, все – объекты, схожие конструкции списков, интервалов, хеш-таблиц. Но, с другой стороны, язык Ruby имеет другое оформление блоков кода. Вместо табуляций используются конструкции “Оператор/end”. Например,

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Пример блока кода Ruby

## Лексический анализатор языка

При разработке парсера таких комплексных языков программирования как Ruby, за определение конкретных структур кода отвечает лексический анализатор.

Например, лексический анализатор определяет следующие ключевые конструкции:

Таблица 1 – Ключевые конструкции

|  |  |
| --- | --- |
| **Название** | **Конструкции** |
| Конструкции, отделяющие часть | begin, end, def, class |
| Операторы и исключения | ”\(?\<\<\~”EOF” “EOF” “class” “module” “def” “undef” “begin” “BEGIN” “END” “Raise” “rescue” “ensure” “end” “if” “unless” “then” “elsif” “else” “case” “when” “while” “until” “for” “break” “next” “redo” “retry” “in” “do” “return” “Yield" “super” “And" “or” “Not" “alias” “self” “true” “False" “nil” “Private" “public” “Command" “lable” “defined” |
| Переменные | (-[ \t]\*)?[:]?[&]?[\*]?[@$a-zA-Z\_]{tIDENTIFIER\_INSIDE}\*[\?!]? |
| Вызов методов | [:][:][\_a-zA-Z]+[\_a-zA-Z0-9]\* |
| Строки | [a-zA-Z]?['][^']\*['] |
| Целые числа | DIGITS ([0-9](\_?[0-9])\*) |
| Вещественные | FLOAT {[0-9](\_?[0-9])\*}{\.{[0-9](\_?[0-9])\*}}{[eE][+-]?{[0-9](\_?[0-9])\*}}|{[0-9](\_?[0-9])\*}{[eE][+-]?{[0-9](\_?[0-9])\*}}|{[0-9](\_?[0-9])\*}{\.{[0-9](\_?[0-9])\*}|{{[0-9](\_?[0-9])\*}\.} |
| Знаки операций | "\*", “=”, “&”, “~", "!", ">", "<", “;”, “.”, “::”, “;”, “?”, “|” , “+=”, “-=”, “\*=”, “/=”, “%=”, “\*\*=”, “&=”, “|=”, “^=”, “<<=”, “>>=”, “&&=”, “||=”, “<=>”, “==”, “===”, “!=”, “>=”, “<=”, “&&”, “||”, “=~”, “!~”, “>>”, “<<” |
| Символы | "," , "(", ")","[","]","{","}",”..”,”...”,”^” |
| Однострочные комментарии | \n\*([ \t]\*#.\*\n)\*[ \t]\*#.\* |
| Массив строк | “%w” |
| Регулярные выражения | ([/].+[/])|(%r=.+=)|(%Q.+) |

Помимо этих конструкций, наш лексический анализатор обрабатывает и другие конструкции, используемые в Ruby. Полученный результат лексический анализатор передает уже синтаксическому анализатору.

Полный код разработанного решения представлен в **Приложении 1**.

## Синтаксический анализатор языка

Грамматика разработана на основе документаций к языку Ruby версии 3.0.

Таблица 2 – Основные синтаксические элементы

|  |  |
| --- | --- |
| **Синтаксический элемент** | **Описание** |
| program | Программа ввода файла программы stmts целиком |
| stmt | Предложение |
| expr | Выражение |
| atom | Наименьший элемент  (первичный )/ Блок команд |
| Call\_args | Вызов аргументов |
| Arg /args | Аргумент / Аргументы |
| litera | Конечный элемент |
| tIDENTIFIER | Идентификатор, который можно использовать как локальную переменную |
| tCONSTANT | Постоянный идентификатор |
| mb\_terms | Символы которыми заканчиваются предложение |

Рубиновая грамматика строится следующим образом:

**program → stmt → arg → expr → atom**

В соответствии с этим мы и начинали выстраивать наш парсер, основываясь на данную схему

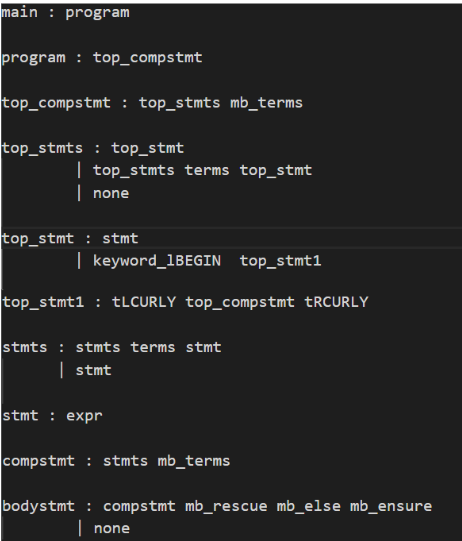


Рисунок 2 – Начало грамматики

«Программа» представляет собой всю грамматику программы. Она эквивалентна «compstmts»(см код), а «compstmts» почти эквивалентна «stmts». `stmts` представляет собой список `stmt`, разделенных `terms`. Следовательно, вся программа - это список `stmt`, разделенных `terms`.

“terms”— это условное название символов, которыми завершаются предложения. Это такие символы, как точка с запятой или перевод строки. А `mb\_terms` в свою очередь, означает «дополнительные условия». None обозначает “ничего”, т. е. используется для возврата указателя `NULL `

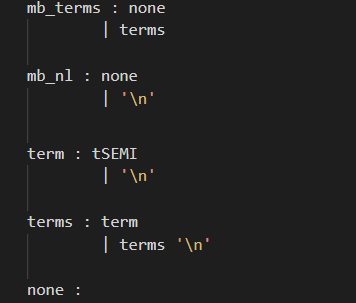


Рисунок 3 – Правила завершения предложений

Stmt состоит из expr, **расписанный в виде дерева, зависящего от ключевых слов, это связано с тем, чтобы избавиться от RR.**

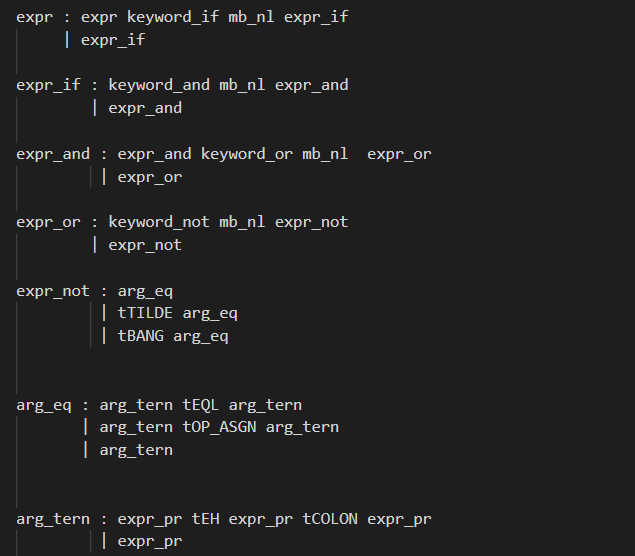


Рисунок 4 – Описание expr

В конечном итоге доходим до правила expr\_pr в котором описаны все случаи использования операций относительно себя и раскрытия правила.

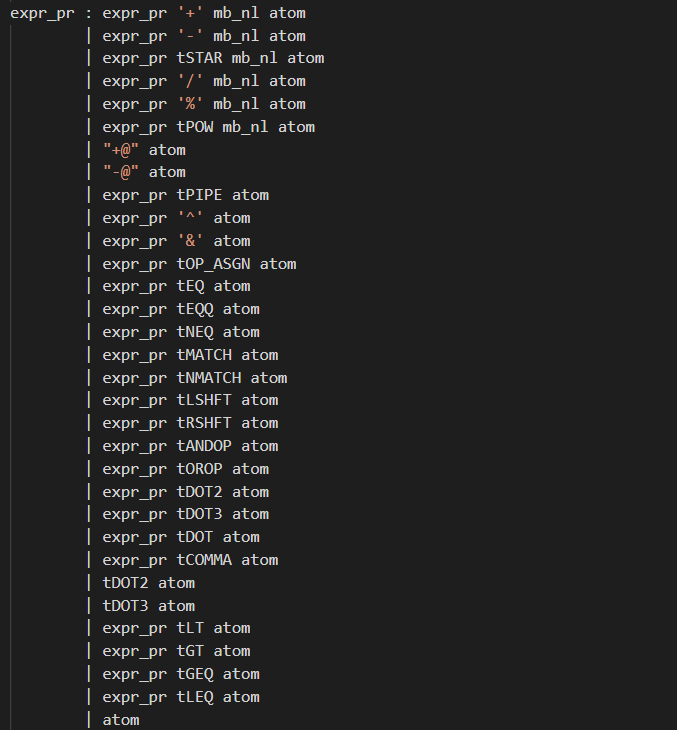


Рисунок 5 – Описание expr\_pr

Остановимся и посмотрим внимательнее на часть “atom”

Atom может состоять из atom\_expr и atom\_tail



Рисунок 6 – Описание правила atom

В atom\_expr описаны все возможные конструкции с ключевыми словами.

В начале обратимся к строчкам:

| tLPAREN compstmt tRPAREN mb\_terms2

| tLBRACK inarray tRBRACK

| tLCURLY assoc\_list tRCURLY

| keyword\_class classpath superclass mb\_terms bodystmt keyword\_end

| keyword\_class tLSHFT expr term bodystmt keyword\_end

| keyword\_module classpath mb\_terms bodystmt keyword\_end

Первая строка позволяет нам обрабатывать синтаксические элементы, взятые в скобки, которые можно передать как аргументы функции (все что угодно). Остальные строки описывают структуры данных, массивы хэши, классы, модули.

Большую часть правила составляют основные конструкции управления и ключевые слова.

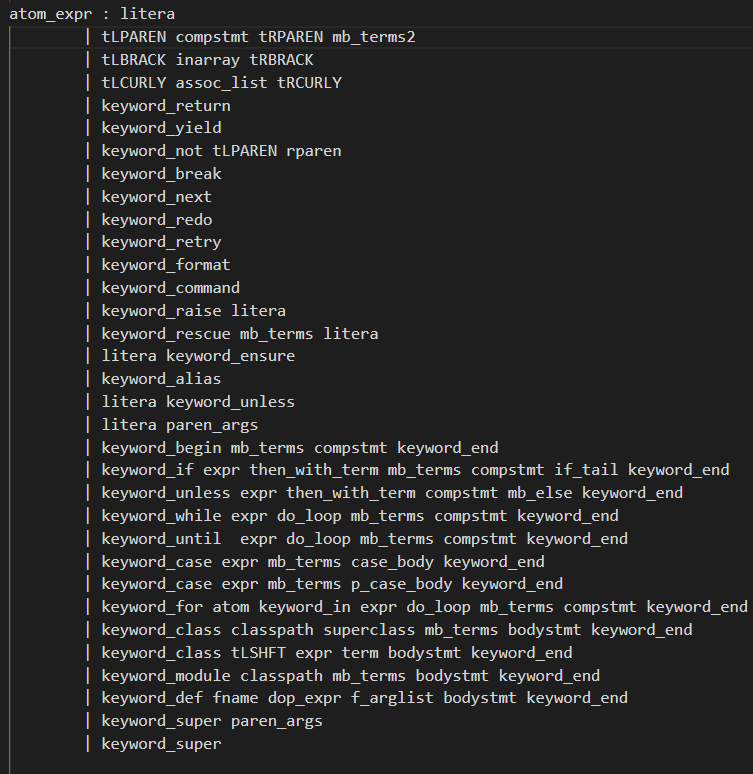


Рисунок 7 – Описание правила atom\_expr

Atom\_tail отвечает за окончание выражений.

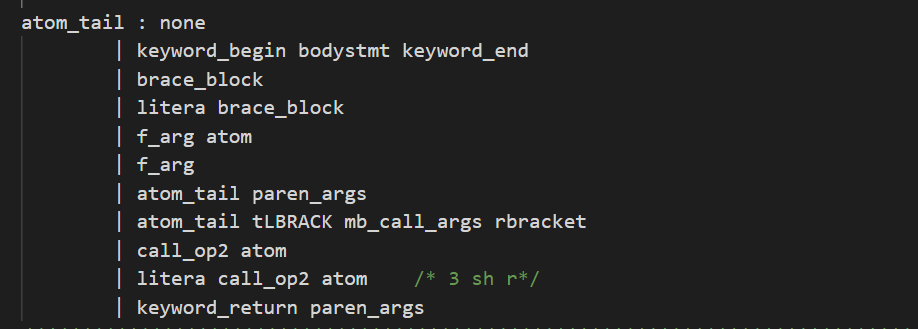


Рисунок 8 – Описание правила atom\_tail

Первая строка отвечает за блок выполнения исключений, который отделен begin и end и вызван из atom\_expr. Вторая и третья строка отвечают за выполнение операций после некоторого условия. Четвертая, пятая, шестая и седьмая строка отвечает за аргументы функции, которые могут быть как в скобках, так и без скобок. Восьмая и девятая – отвечают за вызов методов. Последняя строка отвечает за возврат значений в скобках

Все остальные правила являются вспомогательными в построении основных правил.

## Возникшие конфликты и их исправление

При разработке грамматики парсера языка Ruby возникло много конфликтов. Это возникло из-за того, что язык гибкий и наших знаний не хватает, чтобы сразу описать все правильно. Однако большую часть из них удалось решить.

Изначально вызов функции и методов был описан в отдельном правиле (рисунок 8), причем некоторые из них уже повторялись в atom\_expr и atom\_tail, поэтому было решено отказать от method\_call и расписать правила непосредственно в atom\_expr и atom\_tail. Это привело к тому, что мы избавились от 30конфликтов “Сдвиг/Свертка” и 7 конфликтов “Свертка/Свертка”.

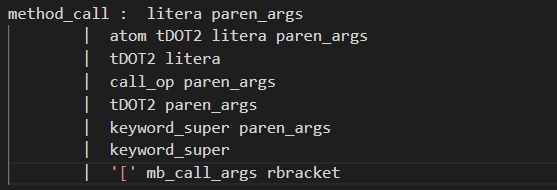


Рисунок 9 – Первичный вариант описания method\_call

В первоначальном варианте atom\_tail paren\_args не был зациклен и пересекался с правилом litera paren\_args в atom\_expr. После добавления цикла мы избавились от 6 конфликтов “Свертка/Свертка”.



Рисунок 10 – Вызов paren\_args

В правиле f\_arg\_item, который необходим для перечисления аргументов в объявлении функции, было описание в виде конечного токена tIDENTIFIER, что вызывало 4 конфликта “Свертка/Свертка” из-за неоднозначности вывода токена tIDENTIFIER, который был уже описан в правиле litera. Поэтому было решено заменить tIDENTIFIER на litera.(Но дало нам 4 конфликта “Сдвиг/Свертка”).

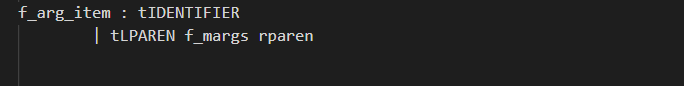


Рисунок 11 – Вызов f\_arg\_item

И того число конфликтов в нашем разработанном парсере равно **43 *конфликта “Сдвиг/Свертка”.***

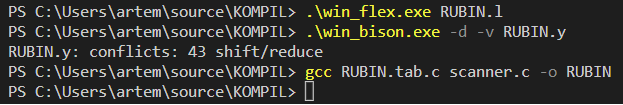
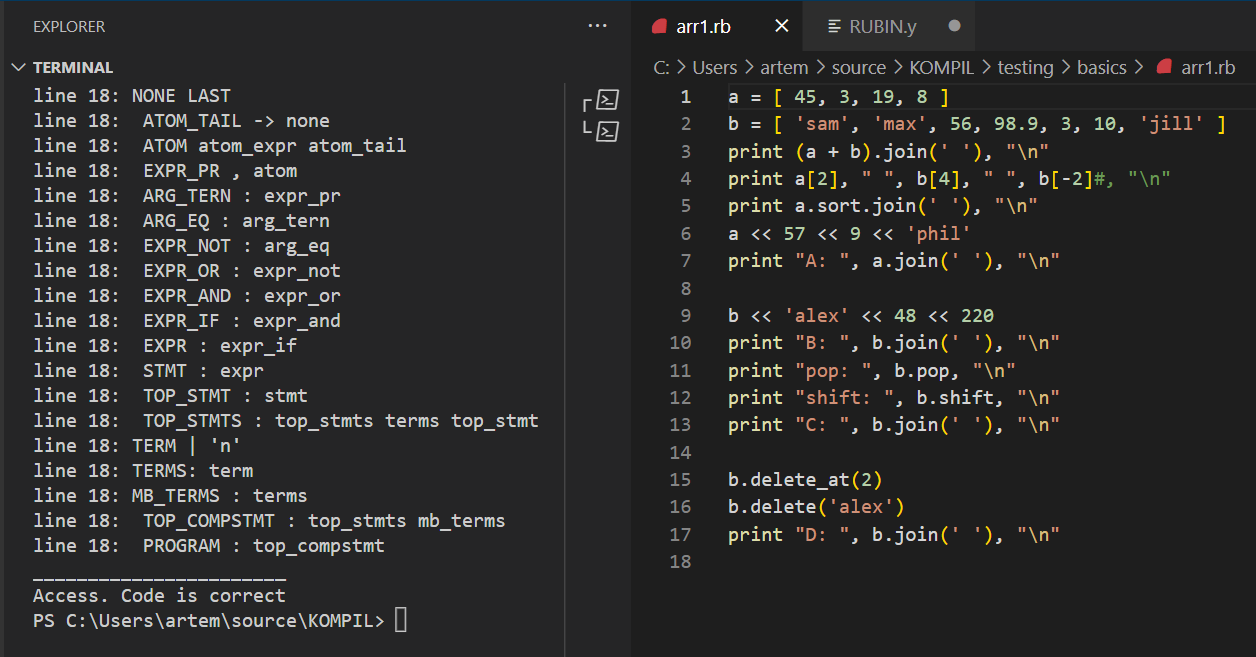
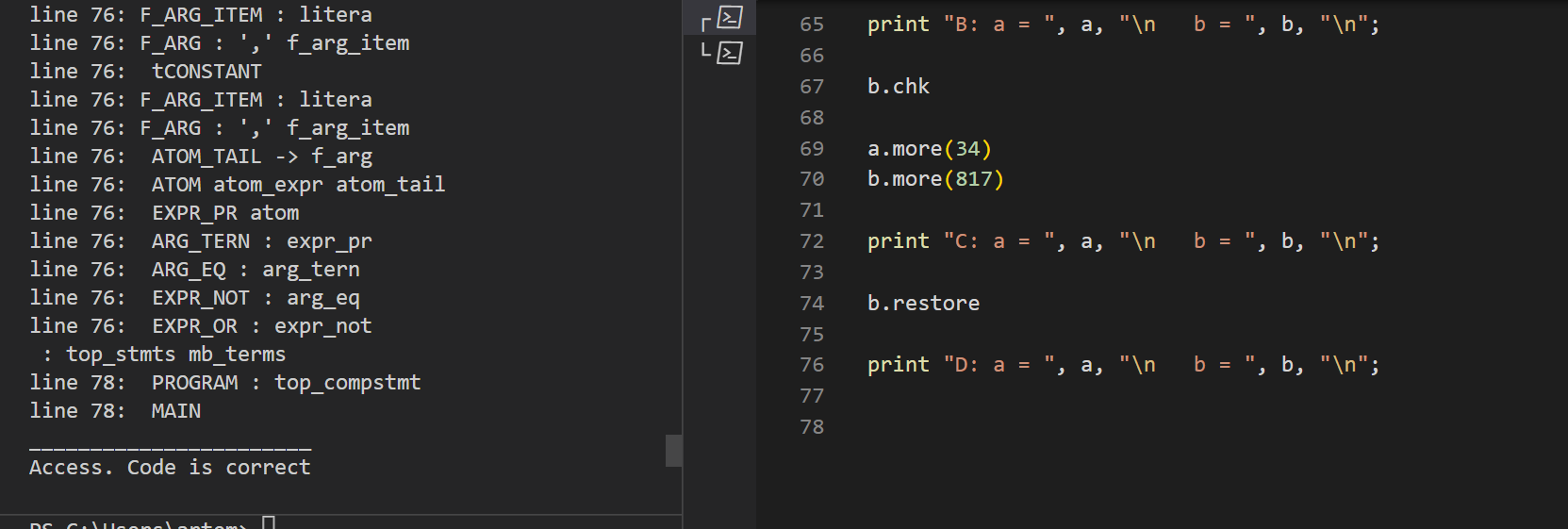


Рисунок 12 – Итоговое количество ошибок при компиляции

## Работоспособность парсера

Проверка работоспособности парсера проводилась на кодах, в которых были описаны основные возможности языка Ruby.

  
Рисунок 13 – Пример работы парсера с массивами

Рисунок 14 – Пример работы парсера с классами

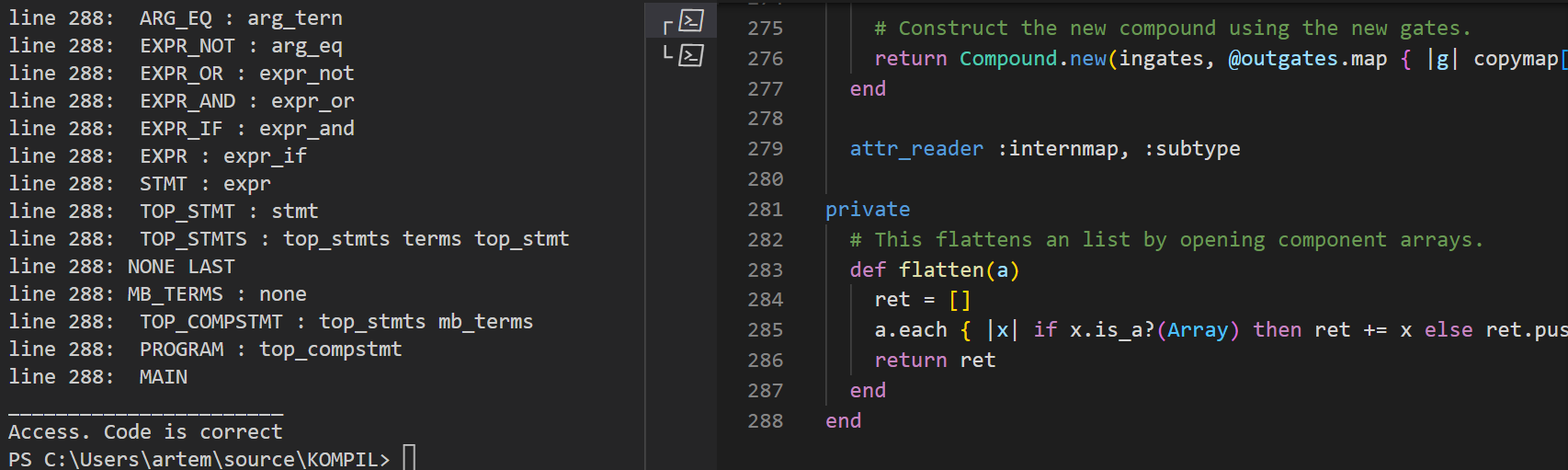
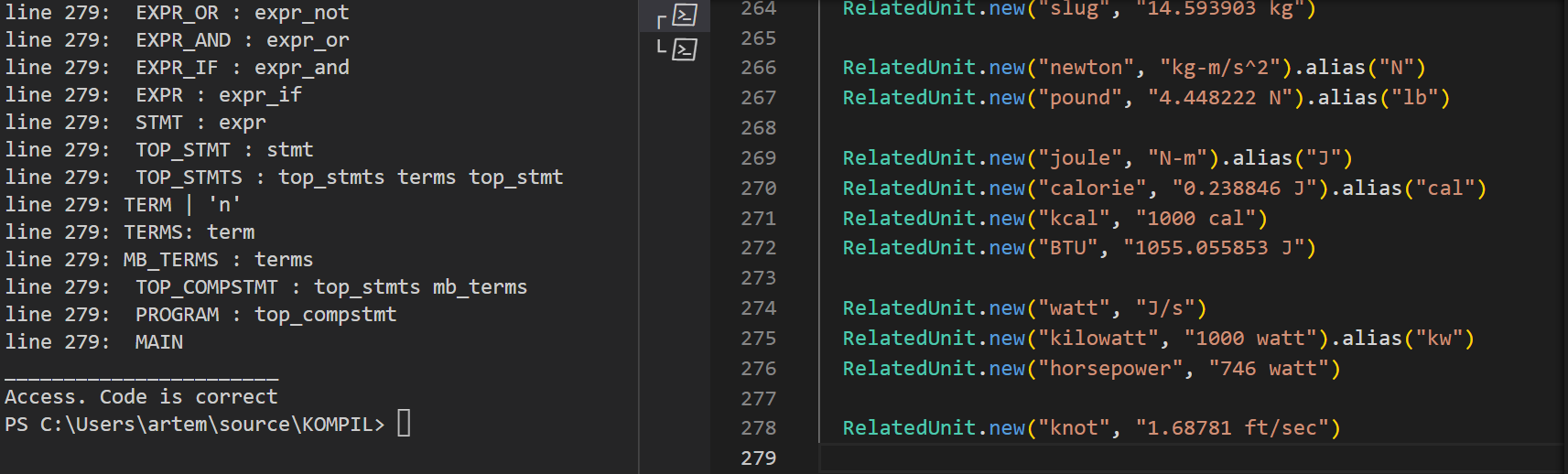


Рисунок 15 – Пример работы парсера с тестовым кодом 1

Рисунок 16 – Пример работы парсера с тестовым кодом 2

**4 РЕЗУЛЬТАТЫ**

В данной курсовой работе были выполнены следующие задачи:

1. Разработан лексический анализатор языка программирования

Ruby.

1. Разрешено большинство конфликтов типа reduce/reducer и shift/reduce в разработанной грамматике.
2. С помощью возможностей bison была собрана программа-

парсер, распознающая код на языке программирования Ruby.

По итогу написания курсовой работы был разработан распознаватель

языка программирования Ruby, обрабатывающий большую часть синтаксиса языка, однако не определяющий некоторые структуры.

В результате, в разработанной программе присутствуют следующие основные «пробелы»:

1. Обработка многострочных строк (heredoc) - Данная

проблема возникает из-за того, что необходимо описывать все варианты конструкции. Она строится следующим образом: символ <<-, затем слово, представляющее имя для данной строки, далее содержимое строки, и закрываем строку словом, которое было в начале, в отдельной строке;

1. Обработка переноса строки в некоторых конструкциях – Даннаяпроблема решаема добавлением terms в нужных местах, однако это приводит к конфликтам “Свертка/Свертка”

# 5 ВЫВОД

Во время выполнения курсовой работы были получены навыки работы со средствами разработки парсеров языков программирования, а также построения грамматик, которые позволят распознать принадлежность входного кода к существующему языку программирования. Был разработан парсер языка Ruby с использованием возможностей средств лексического и синтаксического анализа yacc и flex.

Разработка грамматики основывалась на различных документациях, включая официальную документацию и парсер языка Ruby. Логика языка Ruby была разбита нами на основные части (выражения), что в дальнейшем позволило нам структурированно описать правила грамматики и проработать их.

В связи с тем, что Ruby довольно гибкий язык, который включает себя много уникальных конструкций, сложно создать парсер языка Ruby, который однозначно и полностью распознал бы эти все конструкции. Кроме того, в Ruby существует много дополнительных библиотек – гемов, которые расширяют функционал языка, что делает процесс описания грамматики языка еще сложнее. Следовательно, парсер языка. может быть расширен в зависимости от добавленного гема.

Во время разработки парсера, основной целью было максимально охватить грамматику языка. Но в результате, разработанный парсер языка Ruby имеет конфликты “Сдвиг/Свертка”, что связано с особенностями языка, которые направлены на многофункциональность и удобство использования. Все же данный парсер не обрабатывает некоторые правила, которые мы описали в результатах работы.

Таким образом, исходя из большого количество возникших трудностей и нерешаемых конфликтов при разработке парсера можно сделать вывод о сложности устройства подобных комплексных языков программирования, а также о сложности разработки и структурирования их грамматических и лексических анализаторов. Кроме этого, можно сделать вывод, что с помощью программ yacc/lex либо слишком затруднительно, либо вообще невозможно создать распознаватель языка Ruby, так как даже разработанный распознаватель обладает большим количеством конфликтов и недостатков..

ПРИЛОЖЕНИЕ 1

%{

#include <stdio.h>

#include "parser.h"

extern int yychar;

extern FILE\* yyin;

int yylex();

void yyerror(const char \*s);

int isatty(int fd) { return 0; }

int keyword\_do\_state = 0;

int ignore\_nl = 0;

int ignore\_except\_eof = 0;

static void update\_loc () {

static int curr\_line = 1;

static int curr\_col = 1;

yylloc.first\_line = curr\_line;

yylloc.first\_column = curr\_col;

for (char \*s = yytext; \*s != '\0'; s++) {

if (\*s == '\n') {

curr\_line++;

curr\_col = 1;

} else {

curr\_col++;

}

}

yylloc.last\_line = curr\_line;

yylloc.last\_column = curr\_col-1;

}

#define YY\_USER\_ACTION { update\_loc(); }

%}

%option nounistd yylineno outfile="scanner.c" header="scanner.h"

ESCAPED\_CHAR (\\.|\\\n)

STRING\_QUOTE1 ('([^\\'\n]|{ESCAPED\_CHAR})\*')

STRING\_QUOTE2 (\"([^\\\"\n]|{ESCAPED\_CHAR})\*\")

INSIDE\_LONG\_QUOTE1 ([']{0,2}([^\\']|{ESCAPED\_CHAR}))\*

INSIDE\_LONG\_QUOTE2 ([\"]{0,2}([^\\\"]|{ESCAPED\_CHAR}))\*

LONG\_STRING\_QUOTE1 ('''{INSIDE\_LONG\_QUOTE1}''')

LONG\_STRING\_QUOTE2 (\"\"\"{INSIDE\_LONG\_QUOTE2}\"\"\")

FORMAT\_STRING (\%w)

STRING\_QUOTE ({STRING\_QUOTE1}|{STRING\_QUOTE2}|{LONG\_STRING\_QUOTE1}|{LONG\_STRING\_QUOTE2})

INT10 ([1-9](\_?[0-9])\*)|(0(\_?0)\*)

INT2 0[bB](\_?[01])+

INT8 0[oO](\_?[0-7])+

INT16 0[xX](\_?[0-9a-fA-F])+

INT ({INT10}|{INT2}|{INT8}|{INT16})

DIGITS ([0-9](\_?[0-9])\*)

EXPONENT ([eE][+-]?{DIGITS})

FRACTION \.{DIGITS}

EMPTY\_FRAC {DIGITS}\.

FLOAT {DIGITS}{FRACTION}{EXPONENT}|{DIGITS}{EXPONENT}|{DIGITS}{FRACTION}|{EMPTY\_FRAC}

NUMBER ((-[ \t]\*)?{INT}|{FLOAT})

tIDENTIFIER\_INSIDE [a-zA-Z0-9\_]

%%

\n\*([ \t]\*#.\*\n)\*[ \t]\*#.\* {

// comment: ignore whitespaces

}

\(?\<\<\~EOF { ignore\_except\_eof = 1; }

EOF { ignore\_except\_eof = 0; ignore\_nl--; }

class { return keyword\_class; }

module { return keyword\_module; }

def { return keyword\_def; }

undef { return keyword\_undef; }

begin { return keyword\_begin; }

BEGIN { return keyword\_lBEGIN;}

END { return keyword\_lEND;}

raise { return keyword\_raise;}

rescue { return keyword\_rescue; }

ensure { return keyword\_ensure; }

end { return keyword\_end; }

if { return keyword\_if; }

unless { return keyword\_unless; }

then { return keyword\_then; }

elsif { return keyword\_elsif; }

else { return keyword\_else; }

case { return keyword\_case; }

when { return keyword\_when; }

while { keyword\_do\_state = 1; return keyword\_while; }

until { keyword\_do\_state = 1; return keyword\_until; }

for { keyword\_do\_state = 1; return keyword\_for; }

break { return keyword\_break; }

next { return keyword\_next; }

redo { return keyword\_redo; }

retry { return keyword\_retry; }

in { return keyword\_in; }

do { return keyword\_do\_state == 0 ? keyword\_do : keyword\_do\_loop; }

return { return keyword\_return; }

yield { return keyword\_yield; }

super { return keyword\_super; }

and { return keyword\_and; }

or { return keyword\_or; }

not { return keyword\_not; }

alias { return keyword\_alias; }

self { return keyword\_self; }

true { return keyword\_true; }

false { return keyword\_false;}

nil { return keyword\_nil;}

private { return keyword\_private;}

public { return keyword\_public;}

command { return keyword\_command;}

lable { return keyword\_tLABLE; }

defined { return keyword\_defined; }

"[" { return tLBRACK;}

"]" { return tRBRACK;}

"{" { return tLCURLY;}

"}" { return tRCURLY;}

"," { return tCOMMA; }

"\*\*" { return tPOW; }

"==" { return tEQ; }

"===" { return tEQQ; }

"!=" { return tNEQ; }

">=" { return tGEQ; }

"<=" { return tLEQ; }

"&&" { return tANDOP; }

"||" { return tOROP; }

"=~" { return tMATCH; }

"!~" { return tNMATCH; }

".." { return tDOT2; }

"..." { return tDOT3; }

"<<" { return tLSHFT; }

">>" { return tRSHFT; }

"&." { return tANDDOT; }

"(" { return tLPAREN; }

([/].+[/])|(%r=.+=)|(%Q.+) {return tREG; }

")" { return tRPAREN; }

"=>" { return tASSOC; }

"=" { return tEQL; }

"&" { return tAMPER; }

"\*" { return tSTAR; }

"~" {return tTILDE;}

"!" {return tBANG;}

"<" {return tLT;}

">" {return tGT;}

";" {return tSEMI;}

"." {return tDOT;}

"::" {return tCOLON2;}

":" {return tCOLON;}

"?" {return tEH;}

"|" {return tPIPE;}

"^" {return tCARET;}

"+="|"-="|"\*="|"/="|"%="|"\*\*="|"&="|"|="|"^="|"<<="|">>="|"&&="|"||="|"<=>" {

return tOP\_ASGN;

}

(-[ \t]\*)?[:]?[&]?[\*]?[@$a-zA-Z\_]{tIDENTIFIER\_INSIDE}\*[\?!]? { //[\*]?[:&\*]?[.]? убрали ВЕРНУЛ [:]?[&]?

return tIDENTIFIER;

}

{NUMBER}|({STRING\_QUOTE}[ \t]\*)+ {

return tCONSTANT;

}

"%w" { return keyword\_format; }

,[ \t]\*\n { return ','; }

=[ \t]\*\n { return '='; }

[\\\n](file:///\\\n) { }

\{[ \t]\*\} { return tCONSTANT; }

[ \r\t]+ { }

[ \r\t\n]\*\n {

if (!ignore\_nl){

return '\n';

}

}

. {

return yytext[0];

}

%%

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

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

if (yyin == NULL)

{

printf("File wasn't found\n");

return -1;

}

yyparse();

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("Access. Code is correct\n");

return 0;

}

int yywrap(void)

{

return 1;

}

void print(char\* msg)

{

printf("line %d: %s\n", yylineno, msg);

}

void yyerror(const char \*s) {

printf("Error [%d,%d]: %s\n", yylloc.last\_line, yylloc.last\_column, s);

exit(1);

}

# ПРИЛОЖЕНИЕ 2

%code {

#include <stdio.h>

#include <stdlib.h>

#include "scanner.h"

int yylex();

void yyerror(const char \*s);

extern void print(char\* msg);

extern int keyword\_do\_state;

}

%defines "parser.h"

%locations

%token

keyword\_class keyword\_module keyword\_def keyword\_undef keyword\_begin keyword\_rescue keyword\_ensure keyword\_end keyword\_if keyword\_unless keyword\_then keyword\_elsif keyword\_else keyword\_case keyword\_when keyword\_while keyword\_until keyword\_for keyword\_break keyword\_next keyword\_redo keyword\_retry keyword\_in keyword\_do keyword\_do\_loop keyword\_return keyword\_yield keyword\_super keyword\_and keyword\_or keyword\_not keyword\_alias keyword\_format keyword\_tLABLE keyword\_self keyword\_true keyword\_false keyword\_nil keyword\_lBEGIN keyword\_lEND keyword\_private keyword\_public keyword\_command keyword\_raise keyword\_defined

%token tIDENTIFIER tCONSTANT

%token tPOW tDSTAR

%token tEQ %token tEQQ %token tNEQ %token tGEQ %token tLEQ %token tANDOP %token tOROP %token tMATCH %token tNMATCH %token tDOT2 %token tDOT3 %token tLSHFT %token tRSHFT %token tANDDOT %token tCOLON2 %token tOP\_ASGN %token tASSOC %token tLBRACK %token tSTAR %token tEQL %token tCOMMA tBANG tLT tGT tDOT tEH tCOLON %token tLPAREN tRPAREN tLCURLY tRCURLY tAMPER tRBRACK tTILDE tSEMI tPIPE tREG tCARET

%right tEH tCOLON

%left tLSHFT tRSHFT

%%

main : program

program : top\_compstmt

top\_compstmt : top\_stmts mb\_terms

top\_stmts : top\_stmt

| top\_stmts terms top\_stmt

| none

top\_stmt : stmt

| keyword\_lBEGIN top\_stmt1

top\_stmt1 : tLCURLY top\_compstmt tRCURLY

stmts : stmts terms stmt

| stmt

stmt : expr

compstmt : stmts mb\_terms

bodystmt : compstmt mb\_rescue mb\_else mb\_ensure

| none

expr : expr keyword\_if mb\_nl expr\_if

| expr\_if

expr\_if : keyword\_and mb\_nl expr\_and

| expr\_and

expr\_and : expr\_and keyword\_or mb\_nl expr\_or

| expr\_or

expr\_or : keyword\_not mb\_nl expr\_not

| expr\_not

expr\_not : arg\_eq

| tTILDE arg\_eq

| tBANG arg\_eq

arg\_eq : arg\_tern tEQL arg\_tern

| arg\_tern tOP\_ASGN arg\_tern

| arg\_tern

arg\_tern : expr\_pr tEH expr\_pr tCOLON expr\_pr

| expr\_pr

expr\_pr : expr\_pr '+' mb\_nl atom

| expr\_pr '-' mb\_nl atom

| expr\_pr tSTAR mb\_nl atom

| expr\_pr '/' mb\_nl atom

| expr\_pr '%' mb\_nl atom

| expr\_pr tPOW mb\_nl atom

| "+@" atom

| "-@" atom

| expr\_pr tPIPE atom

| expr\_pr '^' atom

| expr\_pr '&' atom

| expr\_pr tOP\_ASGN atom

| expr\_pr tEQ atom

| expr\_pr tEQQ atom

| expr\_pr tNEQ atom

| expr\_pr tMATCH atom

| expr\_pr tNMATCH atom

| expr\_pr tLSHFT atom

| expr\_pr tRSHFT atom

| expr\_pr tANDOP atom

| expr\_pr tOROP atom

| expr\_pr tDOT2 atom

| expr\_pr tDOT3 atom

| expr\_pr tDOT atom

| expr\_pr tCOMMA atom

| tDOT2 atom

| tDOT3 atom

| expr\_pr tLT atom

| expr\_pr tGT atom

| expr\_pr tGEQ atom

| expr\_pr tLEQ atom

| atom

atom : atom\_expr atom\_tail

litera : tCONSTANT

| tIDENTIFIER

| tOP\_ASGN

| keyword\_variable1

| tREG

atom\_expr : litera

| tLPAREN compstmt tRPAREN mb\_terms2

| tLBRACK inarray tRBRACK

| tLCURLY assoc\_list tRCURLY

| keyword\_return

| keyword\_yield

| keyword\_not tLPAREN rparen

| keyword\_break

| keyword\_next

| keyword\_redo

| keyword\_retry

| keyword\_format

| keyword\_command

| keyword\_raise litera

| keyword\_rescue mb\_terms litera

| litera keyword\_ensure

| keyword\_alias

| litera keyword\_unless

| litera paren\_args

| keyword\_begin mb\_terms compstmt keyword\_end

| keyword\_if expr then\_with\_term mb\_terms compstmt if\_tail keyword\_end

| keyword\_unless expr then\_with\_term compstmt mb\_else keyword\_end

| keyword\_while expr do\_loop mb\_terms compstmt keyword\_end

| keyword\_until expr do\_loop mb\_terms compstmt keyword\_end

| keyword\_case expr mb\_terms case\_body keyword\_end

| keyword\_case expr mb\_terms p\_case\_body keyword\_end

| keyword\_for atom keyword\_in expr do\_loop mb\_terms compstmt keyword\_end

| keyword\_class classpath superclass mb\_terms bodystmt keyword\_end

| keyword\_class tLSHFT expr term bodystmt keyword\_end

| keyword\_module classpath mb\_terms bodystmt keyword\_end

| keyword\_def fname dop\_expr f\_arglist bodystmt keyword\_end

| keyword\_super paren\_args

| keyword\_super

atom\_tail : none

| keyword\_begin bodystmt keyword\_end

| brace\_block

| litera brace\_block

| f\_arg atom

| f\_arg

| atom\_tail paren\_args

| atom\_tail tLBRACK mb\_call\_args rbracket

| call\_op2 atom

| litera call\_op2 atom

| keyword\_return paren\_args

dop\_expr : dop\_expr tDOT litera

| tCOLON2

| none

f\_arglist : f\_arg\_item

| f\_args term

inarray : none

| args trailer

| args tCOMMA assocs trailer

| assocs trailer

paren\_args : tLPAREN mb\_call\_args rparen

| tLPAREN tDOT3 rparen

mb\_call\_args : none

| call\_args

| args tCOMMA

| args tCOMMA assocs tCOMMA

| assocs tCOMMA

call\_args : args mb\_block\_arg

| assocs mb\_block\_arg

| args tCOMMA assocs mb\_block\_arg

| block\_arg

block\_arg : tAMPER expr\_pr

| tAMPER

mb\_block\_arg : tCOMMA block\_arg

| none

args : expr\_pr

| args tCOMMA expr\_pr

mrhs : args tCOMMA expr\_pr

| args tCOMMA tSTAR expr\_pr

| tSTAR expr\_pr

classpath : litera

| litera tCOLON2 litera

| tCOLON2 litera

then\_with\_term : term

| keyword\_then

| term keyword\_then

do\_loop : \_do\_loop {keyword\_do\_state = 0;}

\_do\_loop : term

| keyword\_do\_loop

if\_tail : mb\_else

| keyword\_elsif expr then\_with\_term mb\_terms compstmt if\_tail

mb\_else : none

| keyword\_else mb\_terms compstmt

f\_marg : litera

| tLPAREN f\_margs rparen

| none

f\_marg\_list : f\_marg | f\_marg\_list tCOMMA f\_marg

| f\_marg\_list tEQL f\_marg

f\_margs : f\_marg\_list

| f\_marg\_list tCOMMA f\_rest\_marg | f\_marg\_list tCOMMA f\_rest\_marg tCOMMA f\_marg\_list

| f\_rest\_marg | f\_rest\_marg tCOMMA f\_marg\_list

f\_rest\_marg: tSTAR tIDENTIFIER

| tSTAR

block\_args\_tail : f\_block\_kwarg tCOMMA f\_kwrest mb\_f\_block\_arg

| f\_block\_kwarg mb\_f\_block\_arg | f\_kwrest mb\_f\_block\_arg

| f\_no\_kwarg mb\_f\_block\_arg | f\_block\_arg

mb\_block\_args\_tail : tCOMMA block\_args\_tail

| none

mb\_block\_param : block\_param\_def mb\_nl | none

block\_param\_def : tPIPE mb\_bv\_decl tPIPE

| tPIPE block\_param\_next mb\_bv\_decl tPIPE

block\_param\_next :f\_arg tCOMMA f\_block\_optarg tCOMMA f\_rest\_arg mb\_block\_args\_tail

| f\_arg tCOMMA f\_block\_optarg tCOMMA f\_rest\_arg tCOMMA f\_arg mb\_block\_args\_tail | f\_arg tCOMMA f\_block\_optarg tCOMMA f\_arg mb\_block\_args\_tail

| f\_arg tCOMMA f\_rest\_arg mb\_block\_args\_tail | f\_arg tCOMMA

|f\_arg tCOMMA f\_rest\_arg tCOMMA f\_arg mb\_block\_args\_tail

| f\_arg mb\_block\_args\_tail

| f\_block\_optarg tCOMMA f\_rest\_arg mb\_block\_args\_tail

| f\_block\_optarg tCOMMA f\_rest\_arg tCOMMA f\_arg mb\_block\_args\_tail

| f\_block\_optarg mb\_block\_args\_tail

| f\_block\_optarg tCOMMA f\_arg mb\_block\_args\_tail

| f\_rest\_arg mb\_block\_args\_tail

| f\_rest\_arg tCOMMA f\_arg mb\_block\_args\_tail

|block\_args\_tail

mb\_bv\_decl : mb\_nl

| mb\_nl tSEMI bv\_decls mb\_nl

bv\_decls : tIDENTIFIER

| bv\_decls tCOMMA tIDENTIFIER

brace\_block : keyword\_do mb\_nl do\_body keyword\_end

| tLCURLY mb\_nl brace\_body rbrace

brace\_body : mb\_block\_param compstmt

do\_body : mb\_block\_param bodystmt

case\_args : expr\_pr mb\_terms2

| tSTAR expr\_pr

| case\_args tCOMMA expr\_pr | case\_args tCOMMA tSTAR expr\_pr

case\_body : keyword\_when case\_args then\_with\_term compstmt cases

cases : mb\_else

| case\_body

p\_case\_body : keyword\_in p\_case\_body\_first then\_with\_term mb\_terms compstmt p\_case\_body\_second

p\_case\_body\_first : p\_top\_expr\_body

| p\_top\_expr\_body keyword\_if expr

| p\_top\_expr\_body keyword\_unless expr

p\_top\_expr\_body: p\_as

| p\_as tCOMMA

| p\_as tCOMMA p\_args

| p\_args\_tail

| p\_kwargs

p\_as : p\_as tASSOC tIDENTIFIER

| p\_alt

p\_alt : p\_alt tPIPE p\_expr\_basic | p\_expr\_basic

p\_expr\_basic : p\_expr\_basic\_first

| p\_const tLPAREN p\_args rparen

| p\_const tLPAREN p\_kwargs rparen

| p\_const tLBRACK p\_args rbracket

| p\_const tLBRACK p\_kwargs rbracket | p\_const tLBRACK rbracket

| tLBRACK p\_args rbracket | tLBRACK rbracket

| tLCURLY p\_kwargs rbrace

| tLCURLY rbrace

| tLPAREN p\_as rparen

p\_expr\_basic\_first : p\_primitive

| p\_primitive tDOT2 p\_primitive

| p\_primitive tDOT3 p\_primitive

| p\_primitive tDOT3

| tCARET p\_primitive

| p\_const

| tDOT2 p\_primitive

| tDOT3 p\_primitive

p\_primitive : tIDENTIFIER

p\_args : p\_as

| p\_args\_head

| p\_args\_head p\_as

| p\_args\_head tSTAR tIDENTIFIER

| p\_args\_head tSTAR tIDENTIFIER tCOMMA p\_args\_post | p\_args\_head tSTAR

| p\_args\_head tSTAR tCOMMA p\_args\_post | p\_args\_tail

p\_args\_head : p\_as tCOMMA

| p\_args\_head p\_as tCOMMA

p\_args\_tail : tSTAR tIDENTIFIER

| tSTAR tIDENTIFIER tCOMMA p\_args\_post

| tSTAR

| tSTAR tCOMMA p\_args\_post

p\_args\_post : p\_as

| p\_args\_post tCOMMA p\_as

p\_kwargs : p\_kwarg tCOMMA p\_kwrest

| p\_kwarg

| p\_kwrest

| p\_kwarg tCOMMA p\_kwnorest

| p\_kwnorest

p\_kwarg : p\_kw

| p\_kwarg tCOMMA p\_kw

p\_kw : p\_kw\_label p\_as

| p\_kw\_label

p\_kw\_label : keyword\_tLABLE

p\_kwrest : kwrest\_mark tIDENTIFIER

| kwrest\_mark

p\_kwnorest : kwrest\_mark keyword\_nil

p\_const : tCOLON2 tCONSTANT

| p\_const tCOLON2 tCONSTANT

| tCONSTANT

p\_case\_body\_second: mb\_else | p\_case\_body

mb\_rescue : keyword\_rescue exc\_list exc\_var then\_with\_term mb\_terms compstmt mb\_rescue

| none

exc\_list : expr\_pr

| mrhs

| none

exc\_var : tASSOC lhs

| none

mb\_ensure : keyword\_ensure mb\_terms compstmt

| none

lhs : keyword\_variable1

| atom call\_op tIDENTIFIER

| atom tDOT2 tIDENTIFIER

| atom call\_op tCONSTANT

| atom tDOT2 tCONSTANT

| tCOLON2 tCONSTANT

| none

fname : tIDENTIFIER

| tCONSTANT

| tOP\_ASGN

| reswords

reswords : keyword\_lBEGIN

| keyword\_lEND | keyword\_alias

| keyword\_and

| keyword\_begin

| keyword\_break

| keyword\_case

| keyword\_class

| keyword\_def

| keyword\_defined | keyword\_do

| keyword\_else

| keyword\_elsif

| keyword\_end | keyword\_ensure

| keyword\_true

| keyword\_false

| keyword\_for

| keyword\_in

| keyword\_module

| keyword\_next

| keyword\_nil

| keyword\_not

| keyword\_or

| keyword\_redo

| keyword\_rescue

| keyword\_retry | keyword\_return

| keyword\_self

| keyword\_super

| keyword\_then | keyword\_yield

| keyword\_if

| keyword\_unless

| keyword\_while

| keyword\_until

| keyword\_when

| keyword\_undef | keyword\_command

keyword\_variable1 : keyword\_nil

| keyword\_self

| keyword\_true

| keyword\_false

| keyword\_public

| keyword\_private

args\_tail: f\_kwarg tCOMMA f\_kwrest mb\_f\_block\_arg

| f\_kwarg mb\_f\_block\_arg | f\_kwrest mb\_f\_block\_arg

| f\_no\_kwarg mb\_f\_block\_arg

| f\_block\_arg

mb\_args\_tail : tCOMMA args\_tail

| tEQL args\_tail

| none

f\_args : f\_arg tCOMMA f\_optarg tCOMMA f\_rest\_arg mb\_args\_tail | f\_arg tCOMMA f\_optarg tCOMMA f\_rest\_arg tCOMMA f\_arg mb\_args\_tail

| f\_arg tCOMMA f\_optarg mb\_args\_tail

| f\_arg tCOMMA f\_optarg tCOMMA f\_arg mb\_args\_tail

| f\_arg tCOMMA f\_rest\_arg mb\_args\_tail

| f\_arg tCOMMA f\_rest\_arg tCOMMA f\_arg mb\_args\_tail

| f\_arg mb\_args\_tail

| f\_optarg tCOMMA f\_rest\_arg mb\_args\_tail | f\_optarg tCOMMA f\_rest\_arg tCOMMA f\_arg mb\_args\_tail | f\_optarg mb\_args\_tail

| f\_optarg tCOMMA f\_arg mb\_args\_tail | f\_rest\_arg mb\_args\_tail

| f\_rest\_arg tCOMMA f\_arg mb\_args\_tail | args\_tail

| none

f\_arg\_item : litera

| tLPAREN f\_margs rparen

f\_arg : f\_arg\_item

| f\_arg tCOMMA f\_arg\_item

| f\_arg tDOT f\_arg\_item

f\_kw : keyword\_tLABLE expr\_pr

| keyword\_tLABLE

f\_block\_kw : keyword\_tLABLE atom | keyword\_tLABLE

f\_block\_kwarg : f\_block\_kw

| f\_block\_kwarg tCOMMA f\_block\_kw

f\_kwarg : f\_kw

| f\_kwarg tCOMMA f\_kw

kwrest\_mark : tPOW

f\_no\_kwarg : kwrest\_mark keyword\_nil

f\_kwrest : kwrest\_mark tIDENTIFIER

| kwrest\_mark

f\_opt : tIDENTIFIER tEQ expr\_pr

f\_block\_opt : tIDENTIFIER tEQ atom

f\_block\_optarg : f\_block\_opt

| f\_block\_optarg tCOMMA f\_block\_opt

f\_optarg : f\_opt

| f\_optarg tCOMMA f\_opt

f\_rest\_arg : tSTAR tIDENTIFIER | tSTAR

f\_block\_arg : tAMPER tIDENTIFIER

mb\_f\_block\_arg : tCOMMA f\_block\_arg | none

assoc\_list : none

| assocs trailer

assocs : assoc

| assocs tCOMMA assoc

assoc : expr\_pr tASSOC atom

| keyword\_tLABLE expr\_pr

| tDSTAR expr\_pr

| tPOW | assoc tASSOC expr\_pr

superclass : tLT expr term

| none

call\_op : tDOT

| tANDDOT

call\_op2 : call\_op

| tCOLON2

mb\_terms2 : none

| tCOLON2

| tDOT

| tCOLON

mb\_terms : none

| terms

mb\_nl : none

| '\n'

rparen : mb\_nl tRPAREN

rbracket : mb\_nl tRBRACK

rbrace : mb\_nl tRCURLY

trailer : mb\_nl | tCOMMA

term : tSEMI

| '\n'

terms : term

| terms '\n'

none :