|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт информационных технологий |
| Кафедра инструментального и прикладного программного обеспечения (ИиППО) |
|  |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ** | |
| **по дисциплине** | |
| **«**Теория автоматов и формальных языков**»** | |
|  | |
| Выполнил студент группы ИКБО-16-17 | Акжигитов Р. Р. |
| Принял | Соболев О. В. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «26» ноября 2019 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2019 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва 2019

**Оглавление**

[Цель практической работы 3](#_Toc25653910)

[Задание 3](#_Toc25653911)

[Ход выполнения работы 3](#_Toc25653912)

[Результат исполнения файла 6](#_Toc25653913)

[Дерево разбора 6](#_Toc25653914)

[Заключение 7](#_Toc25653915)

[Список литературных источников 8](#_Toc25653916)

# Цель практической работы

Целью данной практической работы является изучение и разработка лексического и синтаксического анализатора с использования инструмента ANTLR.

# Задание

Входной язык содержит операторы присваивания (:=) и операторы print, оканчивающиеся символом ; (точка с запятой). В левой части оператора присваивания – идентификатор, в правой части – логическое выражение. Логические выражения состоят из идентификаторов, констант 0 и 1, операций or, xor, and, not и круглых скобок. Оператор print имеет аргумент – идентификатор переменной.

// пример программы

x := 1;

y := not y;

print y;

z := (x or y) and not 0;

print z;

# Ход выполнения работы

Лексемы и грамматика (синтаксис) языка описаны в syntax.g4:

// syntax.g4

grammar syntax;

program : line\* EOF ;

line : print\_

| assign

;

logExpr : LPAR logExpr RPAR # parens

| oper=('not' | 'NOT') logExpr # notOp

| logExpr oper=('and' | 'AND') logExpr # andOp

| logExpr oper=('or' | 'OR' | 'xor' | 'XOR') logExpr # orXorOp

| ID # ident

| INT # integer

;

print\_ : PRINT ID COL # printId

| PRINT logExpr COL # printExpr

;

assign : ID ASSIGN logExpr COL # assignExpr;

PRINT: 'PRINT' | 'print' | 'PRT' | 'prt';

ID : [a-zA-Z]+ ; // match identifiers

INT : [0-9]+ ; // match integers

WS : [ \r\n\t]+ -> skip ; // toss out whitespace

LPAR : '(';

RPAR : ')';

ASSIGN : ':=';

COL : ';';

Для получения классов-обработчиков грамматики (listener или visitor) для обхода дерева синтаксического разбора, создания всех дополнительных файлов с токенами и т.д. необходимо сгенерировать файлы с помощью jar пакета ANLTR в необходимый конечный язык:

java -jar .\antlr-4.7.2-complete.jar -Dlanguage=Python3 -visitor syntax.g4

Далее необходимо создать исполняемый файл для считывания входного файла или потока в лексер -> парсер:

// interp.py

import sys

from antlr4 import \*

from antlr4.InputStream import InputStream

from syntaxLexer import syntaxLexer

from syntaxParser import syntaxParser

from MyVisitor import MyVisitor

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) > 1:

input\_stream = FileStream(sys.argv[1])

else:

input\_stream = InputStream(sys.stdin.readline())

lexer = syntaxLexer(input\_stream)

token\_stream = CommonTokenStream(lexer)

parser = syntaxParser(token\_stream)

tree = parser.program()

lisp\_tree\_str = tree.toStringTree(recog=parser)

print(lisp\_tree\_str)

visitor = MyVisitor()

visitor.visit(tree)

Далее необходимо создать собственный класс-посетитель, для обхода по грамматическим конструкциям дерева разбора:

// MyVisitor.py

import sys

from antlr4 import \*

from syntaxVisitor import syntaxVisitor

from syntaxParser import syntaxParser

class MyVisitor(syntaxVisitor):

def \_\_init\_\_(self):

self.memory = {}

def visitAssignExpr(self, ctx):

name = ctx.ID().getText()

value = self.visit(ctx.logExpr())

self.memory[name] = value

return value

def visitPrintExpr(self, ctx):

value = self.visit(ctx.logExpr())

print(value)

return 0

def visitPrintId(self, ctx):

name = ctx.ID().getText()

print(self.memory[name])

# print(self.visit(ctx.id()))

return 0

def visitInteger(self, ctx):

return ctx.INT().getText()

def visitIdent(self, ctx):

name = ctx.ID().getText()

if name in self.memory:

return self.memory[name]

print('[NOTICE]: variable wasn\'t declared, assigned 0 to', name)

self.memory[name] = '0'

# self.memory[name] = input('Input var ' + name)

return self.memory[name]

def visitOrXorOp(self, ctx):

left = int(self.visit(ctx.logExpr(0)))

right = int(self.visit(ctx.logExpr(1)))

if ctx.oper:

if ctx.oper.text.lower() == 'or':

return int(left or right)

elif ctx.oper.text.lower() == 'xor':

return int((left and not right) or (not left and right))

else:

return left

def visitAndOp(self, ctx):

left = int(self.visit(ctx.logExpr(0)))

right = int(self.visit(ctx.logExpr(1)))

if ctx.oper.text.lower() == 'and':

return left and right

else:

return left

def visitNotOp(self, ctx):

left = int(self.visit(ctx.logExpr()))

if ctx.oper.text.lower() == 'not':

return int(not left)

return 0

def visitParens(self, ctx):

return self.visit(ctx.logExpr())

Пример входного файла:

// prog.pr

x := 1;

y := not y;

print y;

z := (x or y) and not 0;

print z;

print 1 xor 0;

print (1 xor ((1 or 1 xor 0 and 1)));

print not 1 xor 0 and 1;

print (not 1 xor 0) and 1;

print not (1 xor 0) and 1;

# Результат исполнения файла

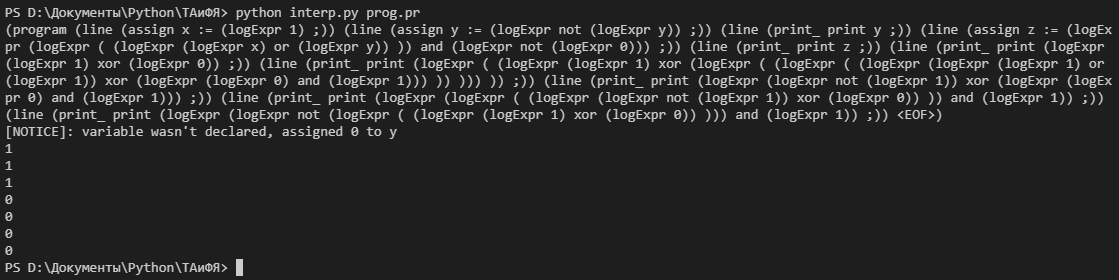


Рисунок 1 – дерево токенов в виде строки и вывод результата

# Дерево разбора

ANTLR в своих классах-обработчиках использует для обхода дерево синтаксического разбора, поэтому для обработки грамматик и визуализации дерева будем использовать именно его.

Файл для запуска графического интерфейса визуализации:

// run.bat

java -jar antlr-4.7.2-complete.jar syntax.g4 -visitor

javac -cp "\*" syntax\*.java

java -cp .;antlr-4.7.2-complete.jar org.antlr.v4.gui.TestRig syntax program -tree -gui prog.pr

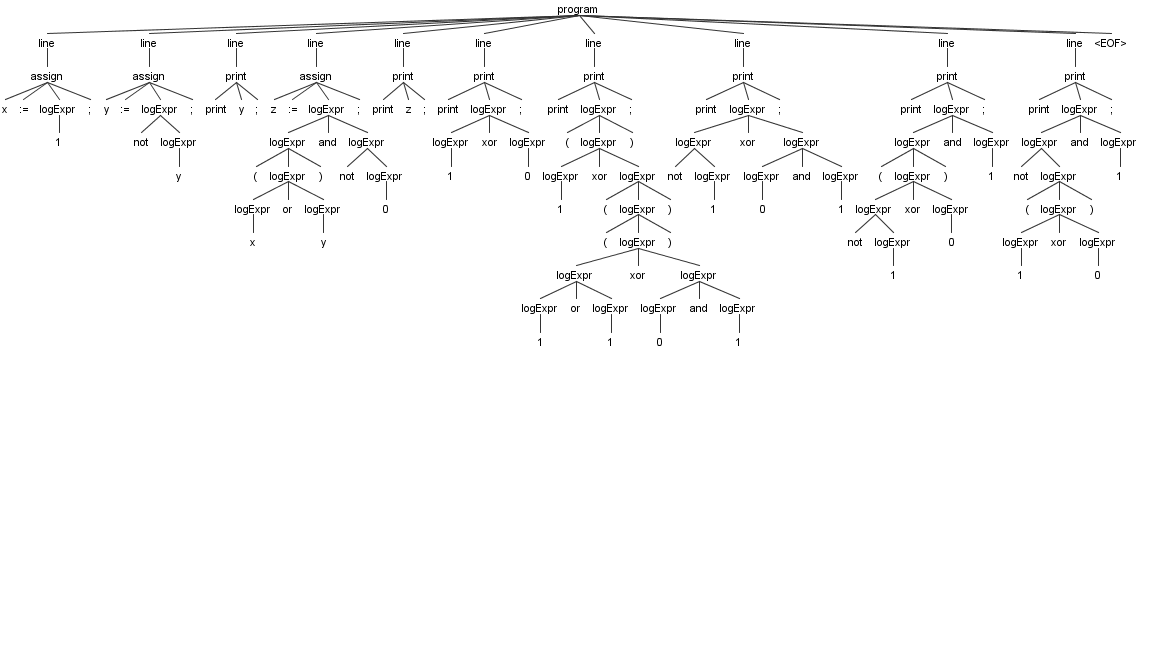


Рисунок 2 – дерево разбора для prog.pr

# Заключение

В данной работе были изучены принципы построения лексического и синтаксического анализатора (состав, принцип работы, способы описания, инструментальные средства для генерации анализаторов на примере ANTLR). Выполнено задание, в котором описана необходимая для варианта грамматика, также описана главная функция для считывания символов из файла и передачи их в лексер и парсер, для последующей обработки грамматики и исполнения файла.

# Список литературных источников

1. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий = Compilers: Principles, Techniques, and Tools. — 2-е изд. — М.: Вильямс, 2008. — ISBN 978-5-8459-1349-4.
2. Робин Хантер. Основные концепции компиляторов = The Essence of Compilers. — М.: «Вильямс», 2002. — С. 256. — ISBN 5-8459-0360-2.
3. Молчанов А.Ю. Системное программное обеспечение: Учебник для вузов. 3-е изд. – СПб.: Питер, 2010 – 400 с.
4. Генератор синтаксических анализаторов, совместимый с YACC для Bison версии 1.35, 25 февраля 2002 [URL: https://www.opennet.ru/docs/RUS/bison\_yacc/]