Министерство образования Республики Беларусь
Учреждения образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №3
По теме «Синтаксический анализатор.»

Выполнил:
студент гр. 053501
Шебеко Ю.А.

Проверил:
Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

1. Цель работы

Освоение работы с существующими синтаксическими анализаторами.

Разработать свой собственный синтаксический анализатор, выбранного подмножества языка программирования. Построить синтаксическое дерево. Определить минимум 4 возможных синтаксических ошибки и показать их корректное выявление. Основной целью работы является написание сценариев, которые задают синтаксические правила для выбранного подмножества языка.

2.Теория

Синтаксический анализатор – часть компилятора, которая отвечает за выявление и проверку синтаксических конструкций входного языка. Синтаксический анализатор получает строку токенов и проверяет, может ли эта строка токенов порождаться грамматикой входного языка.

Результатом синтаксического анализа является синтаксическое строение предложения, представленное либо в виде дерева зависимостей, либо в виде дерева составляющих, либо в виде некоторого сочетания первого и второго способов представления.

Таким образом на основе анализа выражений, состоящих из литералов, операторов и круглых скобок выполняется группирование токенов программы в грамматические фазы, используемые для синтеза вывода.

Типы алгоритмов:

- Нисходящий парсер — продукции грамматики раскрываются, начиная со стартового символа, до получения требуемой последовательности токенов, им соответствуют LL-грамматики;

- Восходящий парсер — продукции восстанавливаются из правых частей, начиная с токенов и кончая стартовым символом, им соответствуют LR-грамматики.

3.Демонстрация работы

Программа написана на Python с использованием библиотеки ANTLR4.

ANTLR — генератор нисходящих анализаторов для формальных языков. ANTLR преобразует контекстно-свободную грамматику в виде расширенная формы Бэкуса — Наура в программу на C++, Java, C#, JavaScript, Swift, Python. Используется для разработки компиляторов, интерпретаторов и трансляторов.

```
stat:  ID ('=' expr)? (simpleOper expr)?';'
    |  'for' '(' type ID ('=' expr)? ' ; ' ID oper expr ' ; ' expr oper (expr)?')' stat
    |  'while' '(' expr ')' stat
    |  type ID (oper expr)? (simpleOper expr)? ';'
    |  'System.out.println' '(' ((vivod)+)? ')' ';'
    |  'return' expr ';'
    |  expr ';'
    |  block
    ;
```

Рисунок 3.1 – вид РФБН

В качестве входных данных рассмотрим задачу о нахождении первых 11 членов последовательности Фибоначчи.

```
(prog (def (funcHeader public (type void) main ( )) (block { (stat (type int) n0 (oper =) (expr 1) ;) (stat (type int) n1 (oper =) (expr 1) ;)
 (stat (type int) n2 ;) (stat System.out.println ( (vivod n0 + " " +) (vivod n1 + " ") ) ;) (stat for ( (type int) i = (expr 3) ; i (oper <=)
 (expr 11) ; (expr i) (oper ++) ) (stat (block { (stat n2 = (expr n0) (simpleOper + ) (expr n1) ;) (stat System.out.println ( (vivod n2 + "
 ") ) ;) (stat n0 = (expr n1) ;) (stat n1 = (expr n2) ;) })))) (stat System.out.println ( ) ;) ))))
```

Рисунок 3.2 – Вывод дерева синтаксического анализа в консоль

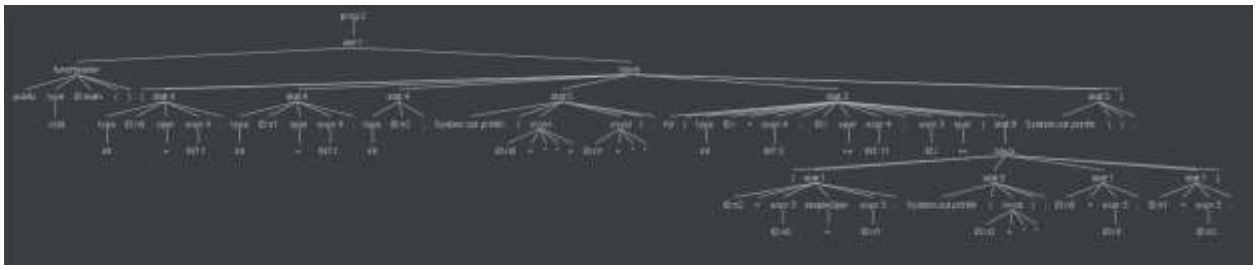

Рисунок 3.3 – Построение дерева синтаксического анализа в ANTLR Preview

```
input.c:2:5 : error: expected expression ";" in line :     int n0 = 1
```

Рисунок 3.4 - Синтаксическая ошибка “;”

На рисунке 3.4 продемонстрирована ошибка отсутствия “;”.

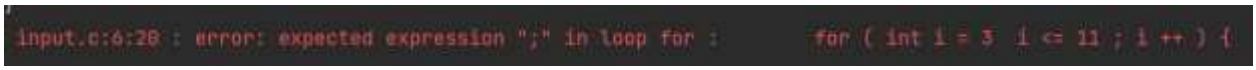

Рисунок 3.5 – Синтаксическая ошибка отсутствие “;” в цикле for

На рисунке 3.5 продемонстрирована синтаксическая ошибка, отсутствия “;” в цикле for.

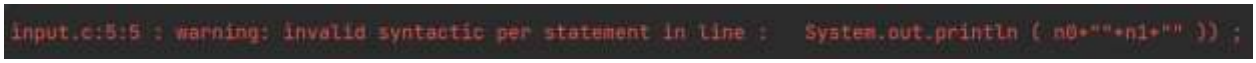

Рисунок 3.6 - Синтаксическая ошибка неверное количество скобок

Рисунок 3.7 – Синтаксическая ошибка присутствует лишняя скобка '}’

Рисунок 3.8 – Синтаксическая ошибка неверное количество кавычек

Приложение А. Текст программы

```
import re
import sys
import antlr4
from antlr4 import *
from gen import *
from gen.treeLexer import treeLexer
from gen.treeParser import treeParser
from gen.treeListener import treeListener

def out_red(text):
    print("\033[31m {}" .format(text))


keyWords = {"while", "for", "if", "else", "int",
            "float", "break", "continue", "double",
            "array", "false", "true"}
loop = {"while", "for"}
operators = {"+", "-", "*", "/", "%",
             "=", ">", "<", ">=", "<=", "==", "!=",
             "++", "--", "+=", "-=", "*=", "/="}
types = {"int ", "float ", "double ", "boolean ", "char ", "string "}
par = {"(", ")", "{", "}"}
prints = {"System.out.println"}
comment = {"//"}
f = open('D://java_not_error.txt', 'r')
text = f.read()
one_word = text.split()
print(text)
f.close()

line = 0
for word in text.split("\n"):
    line = line+1
    s = word.split()
    if s[-1] != ";" and s[-1] != "{" and s[-1] != "}":
        out_red("input.c:%d:%d : error: expected expression in line :
%s" % (line, len(s)+1, word))
        sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line += 1
    pos += 1
    s = word.split()
    if s[0] == "for":
        count = 0
        for x in range(len(s)):
            pos += 1
            if s[x] == ';':
                count += 1
        if count != 2:
            out_red("input.c:%d:%d : error: expected expression \";\" in
loop for :  %s" % (line, pos, word))
            sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line += 1
    pos = 0
    s = list(word)
```

```
    count = 0
    for x in range(len(s)):
        pos += 1
        if s[x] == '\"':
            count += 1
    if count % 2 != 0:
        out_red("input.c:%d:%d : error: expected expression \" in line :
%s" % (line, pos, word))
        sys.exit()

line = 0
pos = 0
rpar = "("
lpar = ")"
count = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    count = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        if letter == rpar:
            count = count + 1
        if letter == lpar:
            count = count - 1
    if count != 0:
        out_red("input.c:%d:%d : warning: invalid syntactic per
statement in line :  %s  " % (line, pos, word))
        sys.exit()

input_stream = antlr4.InputStream(text)
lexer = treeLexer(input_stream)
stream = antlr4.CommonTokenStream(lexer)
parser = treeParser(stream)
tree = parser.prog()
print(tree.toStringTree(recog=parser))
```

Приложение Б. Текст программы

```
grammar tree;

prog:   dec1 | def;

dec1:   funcHeader ';'
    |   type ID ';'
    ;

def :   funcHeader block
    |   type ID ('=' expr)? ';'
    ;

block : '{' stat* '}' ;

stat:  ID ('=' expr)? (simpleOper expr)?';'
    |  'for' '(' type ID ('=' expr)? ' ; ' ID oper expr ' ; ' expr oper
(expr)?')' stat
    |  'while' '(' expr ')' stat
    |  type ID (oper expr)? (simpleOper expr)? ';'
    |  'System.out.println' '(' ((vivod)+)? ')' ';'
    |  'return' expr ';'
    |  expr ';'
    |  block
    ;

expr:   '(' expr ')'
    |   ID '(' expr (',' expr)* ')'
    |   ID
    |   INT
    |   CHAR
    ;

type: 'int' | 'char' | 'void' | 'float' | 'double' | 'String' ;

modifier: 'public' | 'protected' | 'private' ;

vivod: ID'+'((quote)+)?(ID)?((quote)+)?('+')? ;

oper: '<=' | '>=' | '<' | '>' | '+=' | '++' | '--' | '-=' | '=' ;

simpleOper: ' + ' | ' - ' | ' * ' | ' / ' | '+' | '-' | '*' | '/';

quote: ' " ' | '"' | ' "' ;

funcHeader : (modifier)? (' static ')? (type) ID '(' args? ')' ;

args: arg (',' arg)*;

arg : type ID ;

COMMENT : '/*' .*? '*/' -> channel (HIDDEN);

WS : [ \t\n\r]+ -> skip ;

ID : [a-zA-Z_]+ [a-zA-Z0-9_]*;

INT : [0-9]+ ;

CHAR : '\' '~'\''+'\'' ;
```