*Институт Транспорта и связи*



**Лабораторная работа 2**

По дисциплине

«Конструирование Компиляторов»

Тема: **Построение Четвёрок**

Студент: Денис Кобелев

Группа: 4401BD

Рига

2017 г.

1. **Теоретическое обоснование**

***Синтаксис (грамматика)*** – четкие и однозначные грамматические правила, которые должны соблюдаться при написании программы.

***Синтаксический анализатор (парсер)*** – программа для проверки правильности грамматического построения конструкций предложений исходного текста программы.

Задача синтаксического анализатора – распознать предложения исходной программы как языковые конструкции, полностью соответствующие используемой грамматике.

Все конструкции современных языков программирования имеют четкие и однозначные грамматические правила (синтаксис), которые должны соблюдаться при написании программы. Для выявления возможных ошибок и контроля правильности построения предложений языка в исходном тексте необходимо иметь:

1. предложение языка, представленное в виде таблицы стандартных символов
2. грамматику языка, расположенную в памяти (в некоторых методах грамматического разбора это требование может отсутствовать)
3. алгоритм грамматического разбора, реализованный в виде синтаксического анализатора
4. **Грамматические правила для нетерминальных символов**

<if\_statement> ::= if <logical\_condition> then <statement>;

<logical\_condition> ::= (<neq\_condition>) {or (<neq\_condition>)}

<neq\_condition> ::= <val> <> <val>

<statement> ::= ID := <val>

<val> ::= ID | <function\_call>

<function\_call> ::= ID(<parameters>)

<parameters> ::= ID{, ID}

1. **Блок-схема алгоритма**

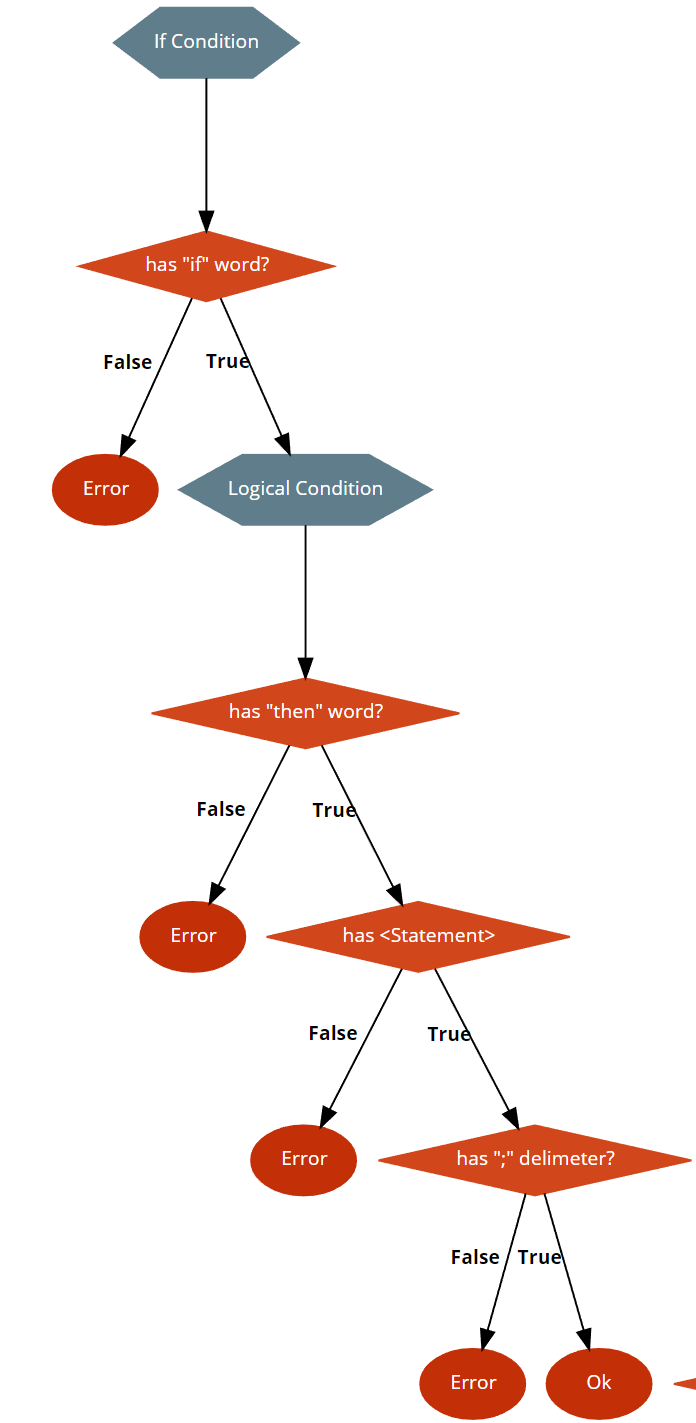


Рис.1. <if\_condition> algorithm

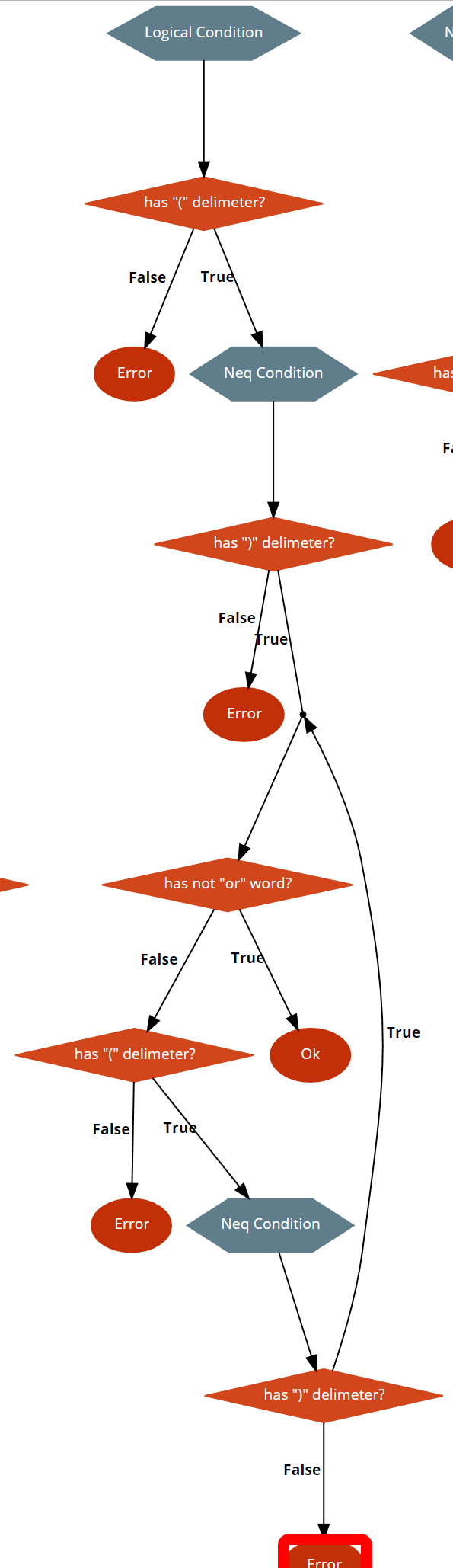


Рис.2. <logical\_condition> algorithm

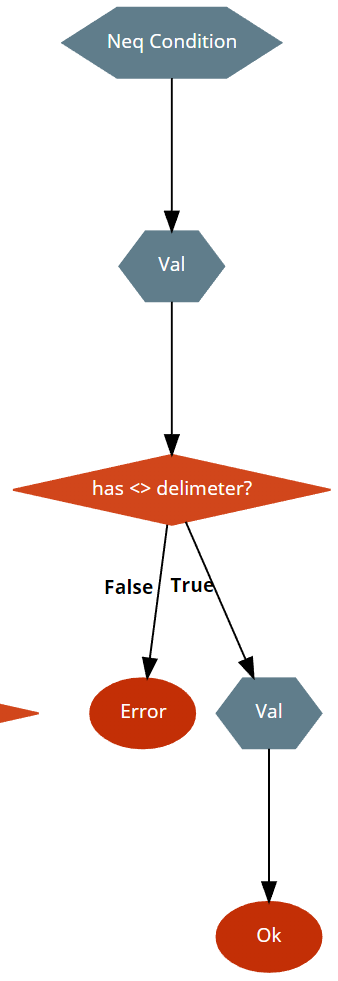


Рис.3. <neq\_condition> algorithm

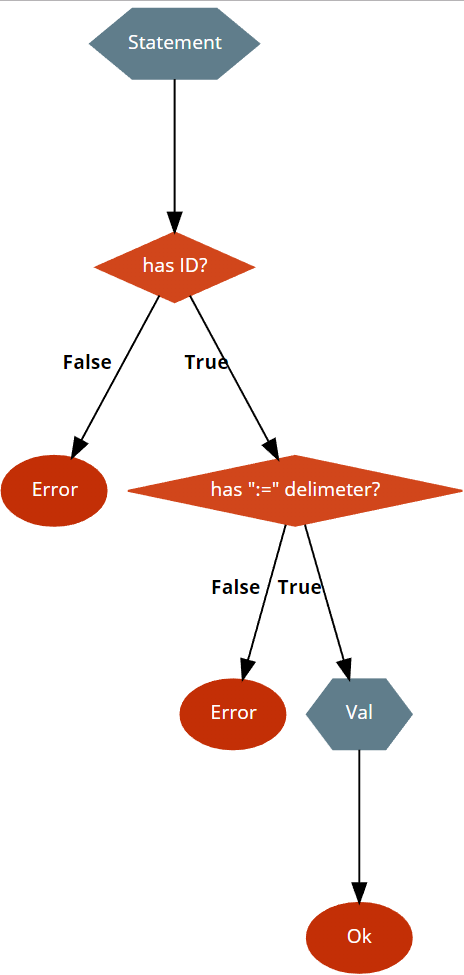


Рис.4. <statement> algorithm

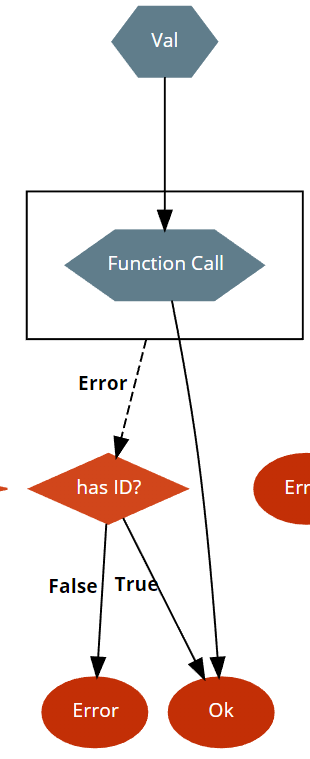


Рис.5. <val> algorithm

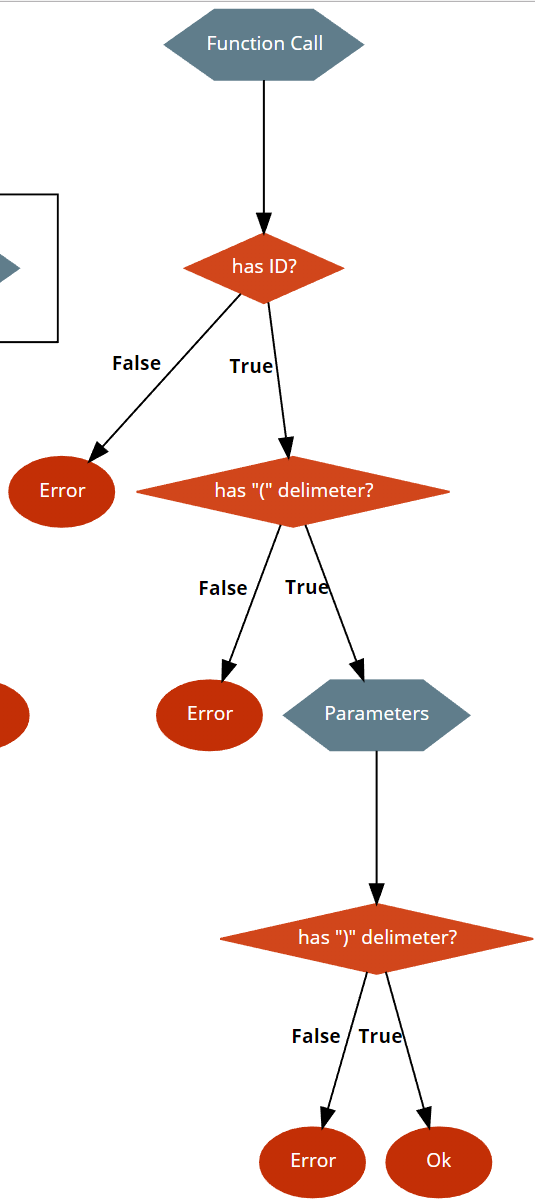


Рис.6. <function\_call> algorithm

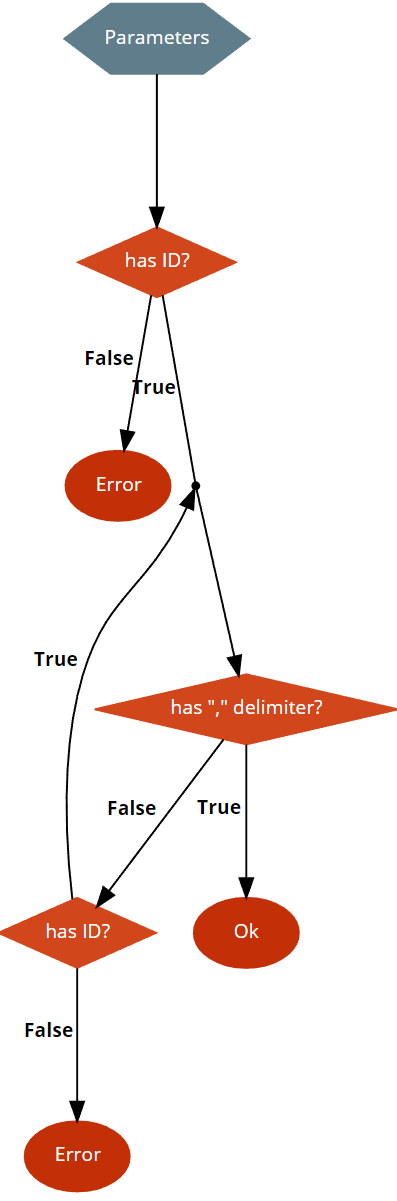


Рис.7. <parameters> algorithm

1. **Текст программы парсера**

class SyntaxValidator(val knownIDs: Set<String>) {  
  
 fun validate(sentence: List<String>): Boolean = ifStatement(sentence).isEmpty()  
  
 fun ifStatement(sentence: List<String>): List<String> {  
 var tail = checkNext("if", sentence)  
 tail = logicalCondition(tail)  
 TetradCollector.makeOperation("BRIFNOT", TetradCollector.nextVariable(), "", "")  
 tail = checkNext("then", tail)  
 tail = statement(tail)  
 return checkNext(";", tail)  
 }  
  
 fun logicalCondition(sentence: List<String>): List<String> {  
 var tail = sentence  
 tail = checkNext("(", tail)  
 tail = neqCondition(tail)  
 var varName = TetradCollector.nextVariable()  
 var resvar = ""  
 tail = checkNext(")", tail)  
 while (tail.*first*() == "or") {  
 tail = tail.*drop*(1)  
 tail = checkNext("(", tail)  
 tail = neqCondition(tail)  
 val varName2 = TetradCollector.nextVariable()  
 resvar = TetradCollector.nextVariable()  
 TetradCollector.makeOperation("OR", varName, varName2, resvar)  
 varName = resvar  
 tail = checkNext(")", tail)  
 }  
 TetradCollector.variable = resvar  
 return tail  
 }  
  
 fun neqCondition(sentence: List<String>): List<String> {  
 var tail = value(sentence)  
 val varName = TetradCollector.nextVariable()  
 tail = checkNext("<>", tail)  
 val result = value(tail)  
 var resvar = ""  
 TetradCollector.makeOperation("EQUALS", varName, TetradCollector.nextVariable(), **{**resvar = TetradCollector.nextVariable(); resvar**}**())  
 TetradCollector.makeOperation("NOT", resvar, "", **{**resvar = TetradCollector.nextVariable(); resvar**}**())  
 TetradCollector.variable = resvar  
 return result  
 }  
  
 fun statement(sentence: List<String>): List<String> {  
 var tail = checkNextIsID(sentence)  
 val varName = TetradCollector.nextVariable()  
 tail = checkNext(":=", tail)  
 val result = value(tail)  
 TetradCollector.makeOperation("ASSIGN", TetradCollector.nextVariable(), "", varName)  
 return result  
 }  
  
 fun value(sentence: List<String>): List<String> {  
 try {  
 return functionCall(sentence)  
 } catch (e: UnexpectedLexemeException) {  
 return checkNextIsID(sentence)  
 }  
 }  
  
 fun functionCall(sentence: List<String>): List<String> {  
 var tail = checkNextIsID(sentence)  
 val varName = TetradCollector.nextVariable()  
 tail = checkNext("(", tail)  
 tail = parameters(tail)  
 val secvar = TetradCollector.nextVariable()  
 val resvar = TetradCollector.nextVariable()  
 TetradCollector.makeOperation("CALL", varName, secvar, resvar)  
 TetradCollector.variable = resvar  
 return checkNext(")", tail)  
 }  
  
 fun parameters(sentence: List<String>): List<String> {  
 var tail = sentence  
 val varName = TetradCollector.nextVariable()  
 tail = checkNextIsID(tail)  
 TetradCollector.makeOperation("SET", TetradCollector.nextVariable(), "", varName)  
 while (tail.*first*() == ",") {  
 tail = tail.*drop*(1)  
 tail = checkNextIsID(tail)  
 TetradCollector.makeOperation("SET", TetradCollector.nextVariable(), "", varName)  
 }  
 TetradCollector.variable = varName  
 return tail  
 }  
  
 fun checkNextIsID(sentence: List<String>): List<String> {  
 val head = sentence.*first*()  
 isID(head)  
 TetradCollector.variable = head  
 return sentence.*drop*(1)  
 }  
  
 fun isID(lexeme: String) {  
 if (!knownIDs.contains(lexeme)) throw UnrecognizedLexemeException(lexeme)  
 }  
  
 fun checkNext(expected: String, sentence: List<String>): List<String> {  
 val head = sentence.*first*()  
 checkLexeme(head, expected)  
 return sentence.*drop*(1)  
 }  
  
 fun checkLexeme(toCheck: String, expected: String) {  
 if (toCheck != expected) throw UnexpectedLexemeException(toCheck, expected)  
 }  
  
 class UnexpectedLexemeException(lexeme: String, expected: String) : Throwable("Unexpected lemexe: $lexeme found when $expected was expected")  
 class UnrecognizedLexemeException(lexeme: String) : Throwable(lexeme + " is not an identifier")  
}

1. **Отпечатанный диагностический файл для исходного программного тескта**
   1. **Без ошибок**

Key Words:   
 0: procedure  
 1: begin  
 2: case  
 3: of  
 4: end  
 5: if  
 6: or  
 7: then  
 8: else  
 9: boolean  
 10: <>  
 11: :=  
 12: (  
 13: )  
 14: :  
 15: ;  
 16: .  
 17: +  
 18: \*  
 19: -  
 20: /  
 21: =  
 22: '  
  
Literals:   
  
Identifiers:   
 0: Longint  
 1: Location  
 2: ALocation  
 3: Modified  
 4: AModified  
  
Result Table:   
 0: if | KEYWORD | 5  
 1: ( | DELIM | 12  
 2: Longint | ID | 0  
 3: ( | DELIM | 12  
 4: Location | ID | 1  
 5: ) | DELIM | 13  
 6: <> | DELIM | 10  
 7: Longint | ID | 0  
 8: ( | DELIM | 12  
 9: ALocation | ID | 2  
 10: ) | DELIM | 13  
 11: ) | DELIM | 13  
 12: or | KEYWORD | 6  
 13: ( | DELIM | 12  
 14: Modified | ID | 3  
 15: <> | DELIM | 10  
 16: AModified | ID | 4  
 17: ) | DELIM | 13  
 18: then | KEYWORD | 7  
 19: Location | ID | 1  
 20: := | DELIM | 11  
 21: ALocation | ID | 2  
 22: ; | DELIM | 15  
  
Statement is correct

* 1. **С сознательно допущенными ошибками**

Key Words:   
 0: procedure  
 1: begin  
 2: case  
 3: of  
 4: end  
 5: if  
 6: or  
 7: then  
 8: else  
 9: boolean  
 10: <>  
 11: :=  
 12: (  
 13: )  
 14: :  
 15: ;  
 16: .  
 17: +  
 18: \*  
 19: -  
 20: /  
 21: =  
 22: '  
  
Literals:   
  
Identifiers:   
 0: iif  
 1: Longint  
 2: Location  
 3: ALocation  
 4: Modified  
 5: AModified  
  
Result Table:   
 0: iif | ID | 0  
 1: ( | DELIM | 12  
 2: Longint | ID | 1  
 3: ( | DELIM | 12  
 4: Location | ID | 2  
 5: ) | DELIM | 13  
 6: <> | DELIM | 10  
 7: Longint | ID | 1  
 8: ( | DELIM | 12  
 9: ALocation | ID | 3  
 10: ) | DELIM | 13  
 11: ) | DELIM | 13  
 12: or | KEYWORD | 6  
 13: ( | DELIM | 12  
 14: Modified | ID | 4  
 15: <> | DELIM | 10  
 16: AModified | ID | 5  
 17: ) | DELIM | 13  
 18: then | KEYWORD | 7  
 19: Location | ID | 2  
 20: := | DELIM | 11  
 21: ALocation | ID | 3  
 22: ; | DELIM | 15  
Unexpected lemexe: iif found when if was expected

1. **Выводы**

В ходе выполнения данной лабораторной работы было необходимо реализовать программу выполняющую Синтаксический анализ для предоставленного выражения на языке Pascal.

В результате была реализована программа выполняющая синтаксический анализ конкретного предложения на языке Pascal на основе предварительного Лексического разбора. Программа была написана на языке Kotlin ввиду личных предпочтений автора и выполняет ввод и вывод в отдельных текстовых файлах.

Так как отчёт составлялся после написания всей серии лабораторных работ по курсу Конструирования Компилятора, в тексте примера анализатора так же находиться код используемый в следующей лабораторной работе, а именно команды составляющие четвёрки.