*Институт Транспорта и связи*



**Лабораторная работа 1**

По дисциплине

«Конструирование Компиляторов»

Тема: **Программирование Синтаксического Анализатора**

Студент: Денис Кобелев

Группа: 4401BD

Рига

2017 г.

1. **Разработанные Четвёрки**

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| SET | Сохранить значение переменной в связку |
| CALL | Вызвать функцию с параметрами из связки |
| EQUALS | Сравнить 2 значения на равенство друг другу |
| NOT | Инвертировать значение |
| OR | Дизъюнкция значений |
| BRIFNOT | Прервать работу, если условие не выполняется |
| ASSIGN | Записать значение в переменную |

1. **Текст разработанной программы**

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. **Результат работы программы**

Tetrads:   
 0: [SET, Location, , T2]  
 1: [CALL, Longint, T2, T3]  
 2: [SET, ALocation, , T4]  
 3: [CALL, Longint, T4, T5]  
 4: [EQUALS, T3, T5, T6]  
 5: [NOT, T6, , T7]  
 6: [EQUALS, Modified, AModified, T8]  
 7: [NOT, T8, , T9]  
 8: [OR, T7, T9, T10]  
 9: [BRIFNOT, T10, , ]  
 10: [ASSIGN, ALocation, , Location]

1. **Выводы**

В ходе выполнения данной лабораторной работы было необходимо написать программу способную составлять серию Тетрад на основе предложения, написанного на языке Pascal.

В результате, было написана программа выполняющая разбиение на четвёрки используя информация предоставляемую Лексическим и Синтаксическим анализаторами. Разбор происходит одновременно с Синтаксическим анализом для повышения эффективности работы алгоритма. Весь код написан на языке Kotlin ввиду личных предпочтений автора.