|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по Лабораторной работе №4**

**по курсу**

**«Проектирование компиляторов»**

**Тема**

**«Синтаксический анализатор операторного предшествования»**

**Вариант 1**

Студент \_\_\_ИУ7-21М\_\_\_\_ \_\_\_\_\_Карпухин А.С.\_\_\_\_

(Группа) (И.О.Фамилия)

Преподаватель \_\_\_\_Ступников А.А.\_\_\_\_

(И.О.Фамилия)

*2021 г.*

**Задание**

**Цель работы:** приобретение практических навыков реализации таблично управляемых синтаксических анализаторов на примере анализатора операторного предшествования.

Задачи работы:

1. Ознакомиться с основными понятиями и определениями, лежащими в основе синтаксического анализа операторного предшествования.
2. Изучить алгоритм синтаксического анализа операторного предшествования.
3. Разработать, тестировать и отладить программу синтаксического анализа в соответствии с предложенным вариантом грамматики.
4. Включить в программу синтаксического анализ семантические действия для реализации синтаксически управляемого перевода инфиксного выражения в обратную польскую нотацию.

**Исходные данные**

В листинге 1 приведена исходная грамматика в соответствии с индивидуальным вариантом. Приведенная грамматика, в которой правые части правил не содержат смежных нетерминалов, представлена в листинге 2.

Листинг 1. Исходная грамматика

|  |
| --- |
| <выражение> ->  <простое выражение> |  <простое выражение> <операция отношения> <простое выражение>  <простое выражение> ->  <терм> |  <знак> <терм> |  <простое выражение> <операция типа сложения> <терм>  <терм> ->  <фактор> |  <терм> <операция типа умножения> <фактор>  <фактор> ->  <идентификатор> |  <константа> |  ( < простое выражение > ) |  not <фактор>  <операция отношения> ->  == | <> | < | <= | > | >=  <знак> ->  + | -  <операция типа сложения> ->  + | - | or  <операция типа умножения> ->  \* | / | div | mod | and  <идентификатор> -> “[\_a-zA-Z][\_a-zA-Z0-9]{0,30}”  <константа> -> “-?[0-9]+|\".\*\"|\'.\*\'” |

Листинг 2. Приведенная грамматика без смежных нетерминалов

|  |
| --- |
| <выражение> ->  <простое выражение> |  <простое выражение> == <простое выражение>  <простое выражение> <> <простое выражение>  <простое выражение> < <простое выражение>  <простое выражение> <= <простое выражение>  <простое выражение> > <простое выражение>  <простое выражение> >= <простое выражение>  <простое выражение> ->  <терм> |  + <терм> |  - <терм> |  <простое выражение> + <терм>  <простое выражение> - <терм>  <простое выражение> or <терм>  <терм> ->  <фактор> |  <терм> \* <фактор>  <терм> / <фактор>  <терм> div <фактор>  <терм> mod <фактор>  <терм> and <фактор>  <фактор> ->  <идентификатор> |  <константа> |  ( < простое выражение > ) |  not <фактор>  <идентификатор> -> “[\_a-zA-Z][\_a-zA-Z0-9]{0,30}”  <константа> -> “-?[0-9]+|\".\*\"|\'.\*\'” |

**Листинги кода**

Код генератора таблицы операторного предшествования на основе исходной грамматики приведен в листинге 1. В листинге 2 приведена реализация синтаксического анализатора операторного предшествования.

Листинг 1. Генерация таблицы операторного предшествования

|  |
| --- |
| package analysis  import grammar.\_  import scala.collection.immutable.SortedSet  import PrecedenceType.\_  /\*\* Specifies control table for table-driven    \* syntax analysis algorithm of operator precedence.    \* @param grammar Operator precedence grammar.    \*/  class ControlTable(var grammar: Grammar) {    val boarder = new GrammarSymbol("$")    val axiom = new GrammarSymbol("axiom")    val table: Map[(GrammarSymbol, GrammarSymbol), PrecedenceType] =      buildControlTable()    /\*\* Build control table for all terminal symbols based on grammar data.      \* @return Raw representation of control table (map).      \*/    private def buildControlTable()        : Map[(GrammarSymbol, GrammarSymbol), PrecedenceType] = {      /\*\* Get terminal symbol following specified terminal.        \* @param term Terminal ыньищд to search for next.        \* @param rule Grammar rule that presumably contains required symbols.        \* @return Required terminal symbol or none if not found.        \*/      def getNextTerm(term: GrammarSymbol)(rule: GrammarRule): Option[GrammarSymbol] = {        if (!rule.rhs.contains(term)) return None        val ind = rule.rhs.indexOf(term)        return if (          rule.rhs.isDefinedAt(ind + 1) &&          rule.rhs(ind + 1).stype == SymbolType.Term        ) Option(rule.rhs(ind + 1))        else if (          rule.rhs.isDefinedAt(ind + 2) &&          rule.rhs(ind + 2).stype == SymbolType.Term        ) Option(rule.rhs(ind + 2))        else None      }      /\*\* Get nonterminal symbol following or precedes specified terminal.        \*        \* @param term Terminal symbol to search for next.        \* @param rule Grammar rule that presumably contains required symbols.        \* @return Required nonterminal symbol or none if not found.        \*/      def getNonterm(          term: GrammarSymbol,          offset: Int      )(rule: GrammarRule): Option[GrammarSymbol] = {        if (!rule.rhs.contains(term)) return None        val ind = rule.rhs.indexOf(term)        return if (          rule.rhs.isDefinedAt(ind + offset) &&          rule.rhs(ind + offset).stype == SymbolType.NonTerm        ) Option(rule.rhs(ind + offset))        else None      }      /\*      grammar = new Grammar(        this.grammar.terms + boarder,        this.grammar.nonTerms + axiom,        this.grammar.rules + new GrammarRule(          axiom :: Nil,          boarder :: this.grammar.axiom :: boarder :: Nil        ),        axiom      )       \*/      val extremeTable = buildExtremeTerminalTable()      var controlTable =        Map[(GrammarSymbol, GrammarSymbol), PrecedenceType.PrecedenceType]()      for (term <- grammar.terms) {        val neighborSymbols = grammar.rules.map(getNextTerm(term) \_).flatten        // Step 1: fill in table cells for symbols that have common basis with current        for (symbol <- neighborSymbols) {          controlTable += ((term, symbol) -> PrecedenceType.Neighbors)        }        // Step 2: fill in table cells for symbols that precedes current in some basis        val leftmost = grammar.rules.map(getNonterm(term, 1) \_).flatten        controlTable ++= leftmost          .map(nt => extremeTable(nt).\_1)          .flatten          .filter(s => s.stype == SymbolType.Term)          .map(s => (term, s) -> PrecedenceType.Precedes)        // Step 3: fill in table cells for symbols that follows current in some basis        val rightmost = grammar.rules.map(getNonterm(term, -1) \_).flatten        controlTable ++= rightmost          .map(nt => extremeTable(nt).\_2)          .flatten          .filter(s => s.stype == SymbolType.Term)          .map(s => (s, term) -> PrecedenceType.Follows)        /\*\* Step 4: fill in table cells for start and stop symbol          \*      if (extremeTable.contains(term)) {          \*        controlTable ++= extremeTable(term).\_1          \*          .filter(s => s.stype == SymbolType.Term)          \*          .map(s => ((start, s) -> PrecedenceType.Precedes))          \*          \*        controlTable ++= extremeTable(term).\_2          \*          .filter(s => s.stype == SymbolType.Term)          \*          .map(s => ((s, stop) -> PrecedenceType.Follows))          \*      }          \*/      }      controlTable ++= extremeTable(grammar.axiom).\_1        .filter(s => s.stype == SymbolType.Term)        .map(s => ((boarder, s) -> PrecedenceType.Precedes))      controlTable ++= extremeTable(grammar.axiom).\_2        .filter(s => s.stype == SymbolType.Term)        .map(s => ((s, boarder) -> PrecedenceType.Follows))      return controlTable    }    /\*\* Build extreme (leftmost and rightmost) nonterminal symbols table.      \* @return Extreme nonterminal symbols table.      \*/    def buildExtremeNonterminalTable()        : Map[GrammarSymbol, (Set[GrammarSymbol], Set[GrammarSymbol])] = {      var table = Map[GrammarSymbol, (Set[GrammarSymbol], Set[GrammarSymbol])]()      for (nt <- grammar.nonTerms) {        val rules = grammar.rules.filter(r => r.lhs.contains(nt))        val pair =          (rules.map(r => r.rhs.head).toSet, rules.map(r => r.rhs.last).toSet)        table += (nt -> pair)      }      var isChanged = true      while (isChanged) {        isChanged = false        for ((symbol, sets) <- table) {          val leftmost = sets.\_1            .filter(s => table.contains(s))            .map(s => table(s).\_1)            .flatten ++ sets.\_1          val rightmost = sets.\_2            .filter(s => table.contains(s))            .map(s => table(s).\_2)            .flatten ++ sets.\_2          table += (symbol -> (leftmost, rightmost))          if (leftmost != sets.\_1 || rightmost != sets.\_2) {            isChanged = true          }        }      }      return table    }    /\*\* Build extreme (leftmost and rightmost) terminal symbols table.      \* @return Extreme terminal symbols table.      \*/    def buildExtremeTerminalTable()        : Map[GrammarSymbol, (Set[GrammarSymbol], Set[GrammarSymbol])] = {      val table = buildExtremeNonterminalTable()      var termTable =        Map[GrammarSymbol, (Set[GrammarSymbol], Set[GrammarSymbol])]()      for (nt <- grammar.nonTerms) {        val rules = grammar.rules.filter(r => r.lhs.contains(nt))        val leftmost = rules          .filter(r =>            r.rhs.head.stype == SymbolType.Term ||              (r.rhs.isDefinedAt(1) &&                r.rhs.tail.head.stype == SymbolType.Term)          )          .map(r =>            if (r.rhs.head.stype == SymbolType.Term) r.rhs.head            else r.rhs.tail.head          )        val rightmost = rules          .filter(r =>            r.rhs.last.stype == SymbolType.Term ||              (r.rhs.isDefinedAt(r.rhs.length - 2) &&                r.rhs(r.rhs.length - 2).stype == SymbolType.Term)          )          .map(r =>            if (r.rhs.last.stype == SymbolType.Term) r.rhs.last            else r.rhs(r.rhs.length - 2)          )        termTable += (nt -> (leftmost, rightmost))      }      for ((symbol, sets) <- table) {        val leftmost = sets.\_1          .filter(s => termTable.contains(s))          .map(s => termTable(s).\_1)          .flatten ++ termTable(symbol).\_1        val rightmost = sets.\_2          .filter(s => termTable.contains(s))          .map(s => termTable(s).\_2)          .flatten ++ termTable(symbol).\_2        termTable += (symbol -> (leftmost, rightmost))      }      return termTable    }    override def toString(): String = {      val (rows, cols) = {        val (r, c) = table.keys.unzip        (SortedSet(r.toSeq: \_\*).toList, SortedSet(c.toSeq: \_\*).toList)      }      ("      " +: cols).map("%5s |".format(\_)).mkString + "\n" +        rows          .map { r =>            "-" + "------|" \* (cols.length + 1) + "\n" +              "%5s  |".format(r.name) + cols.map { c =>                "%4s  |".format(table.getOrElse((r, c), "--"))              }.mkString          }          .mkString("\n")    }  } |

Листинг 2. Код алгоритма синтаксического анализа

|  |
| --- |
| package analysis  import grammar.\_  import scala.collection.mutable.Stack  import java.{util => ju}  class Parser(val grammar: Grammar, val table: ControlTable) {    def parse(input: List[GrammarSymbol]): List[GrammarSymbol] = {      val stack = Stack[GrammarSymbol](table.boarder)      val chain = input :+ table.boarder      var pos = 0      var postfix = List[GrammarSymbol]()      def matchRule(rhs: List[GrammarSymbol])(rule: GrammarRule): Boolean = {        val f1 = rhs.filter(s => s.stype == SymbolType.Term)        val f2 = rule.rhs.filter(s => s.stype == SymbolType.Term)        return f1 == f2      }      def shift(): Unit = {        stack.push(chain(pos))        pos += 1      }      def conv(): Unit = {        var rhs = stack.pop() :: Nil        while (          stack.top.stype == SymbolType.NonTerm ||          !rhs.find(s => s.stype == SymbolType.Term).isDefined ||          table.table(            stack.top,            (rhs.find(s => s.stype == SymbolType.Term).get)          ) == PrecedenceType.Neighbors        ) {          rhs = stack.pop() +: rhs        }        val rule = grammar.rules.find(matchRule(rhs) \_)        if (!rule.isDefined) {          throw new NoSuchElementException(s"No matching rule for base ${rhs}")        }        postfix = postfix ::: rhs.filter(s =>          s.stype == SymbolType.Term && !"()".contains(s.name)        )        stack.push(rule.get.lhs.head)      }      while (true) {        val s = stack.find(s => s.stype == SymbolType.Term).get        if (s == table.boarder && chain(pos) == table.boarder) return postfix        table.table((s, chain(pos))) match {          case PrecedenceType.Neighbors | PrecedenceType.Precedes => shift()          case PrecedenceType.Follows                             => conv()        }      }      return postfix    }  } |

**Результаты работы программы**

Таблица операторного предшествования, генерируемая разработанной программой для грамматики индивидуального варианта, приведена на рисунке 1.

В листинге 3 приведены примеры работы программы для различных входных данных.

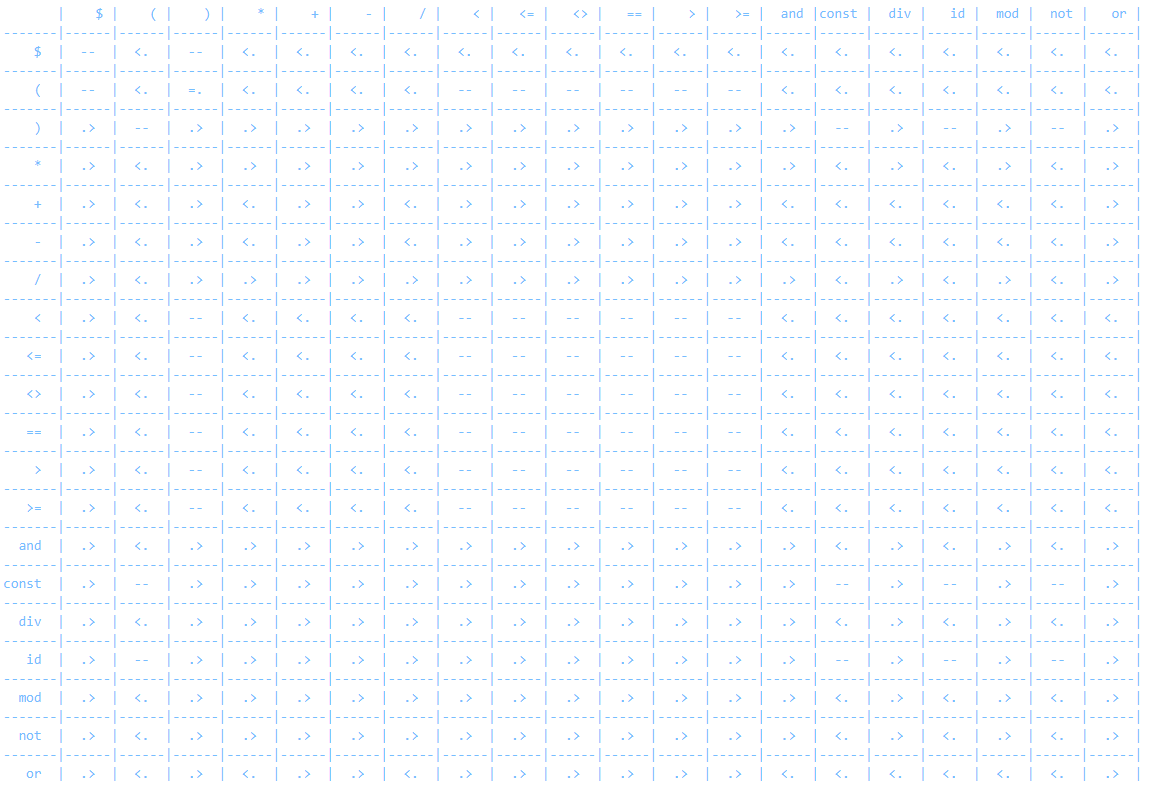
****

Рисунок 1 – Генерируемая программой таблица операторного предшествования для грамматики индивидуального варианта

Листинг 3. Примеры работы программы

|  |
| --- |
| Input: "str" + 56 - ( 78 or ( + 12 ) + 22 ) \* 11  Output: "str" 56 + 78 12 + or 22 + 11 \* -  Input: "str" + 56 - ( 78 or 12 \* 22 ) \* 11  Output: "str" 56 + 78 12 22 \* or 11 \* -  Input: "str" + ( 56 - ( ( 78 or 12 ) \* 22 ) ) \* 11  Output: "str" 56 78 12 or 22 \* - 11 \* + |

**Заключение**

В ходе выполнения работы были изучены методы таблично управляемого синтаксического анализа на примере анализатора операторного предшествования.

Исходная грамматика индивидуального варианта была преобразована к форме без смежных нетерминалов в правых частях правил.

Для полученной грамматики был реализован генератор управляющей таблицы операторного предшествования.

Для сформированных исходных данных была реализована программа восходящего синтаксического анализатора операторного предшествования, примеры работы которой были приведены выше.