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»

Лабораторная работа № 3

Дисциплина Конструирование компиляторов

Тема Синтаксический разбор с использованием метода рекурсивного спуска

Вариант №02

Студент Кибамба Ж.Ж.

Группа ИУ7И-11М

Преподаватель Ступников А.А.

22.05.2024

Москва, 2024 г.

Описание задания

Варианта №2 – Грамматика G2.

Рассматривается грамматика выражений отношения с правилами

<выражение> ->

<арифметическое выражение> <операция отношения> <арифметическое выражение> |

<арифметическое выражение>

<арифметическое выражение> ->

<арифметическое выражение> <операция типа сложения> <терм> |

<терм>

<терм> ->

<терм> <операция типа умножения> <фактор> |

<фактор>

<фактор> ->

<идентификатор> |

<константа> |

(<арифметическое выражение>)

<операция отношения> ->

< | <= | = | <> | > | >=

<операция типа сложения> ->

+ | -

<операция типа умножения> ->

* | /

Замечания.

1. Нетерминалы <идентификатор> и <константа> - это лексические единицы (лексемы), которые оставлены неопределенными, а при выполнении лабораторной работы можно либо рассматривать их как терминальные символы, либо определить их по своему усмотрению и добавить эти определения.

2. Терминалы () - это разделители и символы пунктуации.

3. Терминалы < <= = <> > >= + - * / - это знаки операций.

4. Нетерминал <выражение> - это начальный символ грамматики.

Задание на лабораторную работу

Дополнить грамматику блоком, состоящим из последовательности операторов присваивания. Для реализации предлагаются два варианта расширенной грамматики.

Вариант в стиле Алгол-Паскаль.

<программа> ->

<блок>

<блок> ->

begin<список операторов>end

<список операторов> ->

<оператор> | <список операторов>; <оператор>

<оператор> ->

<идентификатор> = <выражение>

Вариант в стиле Си.

<программа> ->

<блок>

<блок> ->

{<список операторов>}

<список операторов>

<оператор> <хвост>

<хвост> ->

; <оператор> <хвост> | ε

Первый вариант содержит левую рекурсию, которая должна быть устранена. Второй вариант не содержит левую рекурсию, но имеет ε-правило. В обоих вариантах точка с запятой (;) ставится между операторами. Теперь начальным символом грамматики становится нетерминал <программа>. Оба варианта содержат цепное правило <программа> -> <блок>. Можно начальным символом грамматики назначить нетерминал <блок>. А можно <блок> считать оператором, т. е.

<оператор> ->

<идентификатор> = <выражение> |

<блок>

В последнем случае возможна конструкция с вложенными блоками. Если между символом присваивания (=) и символом операции отношения (=) возникает конфликт, то можно для любого из них ввести новое изображение, например,:=, <-, == и т. п.

Для модифицированной грамматики написать программу нисходящего синтаксического анализа с использованием метода рекурсивного спуска.

Решение

Используем первый вариант – в стиле Алголь-Паскаль, и назначаем нетерминал <блок> начальным символом грамматики, тогда исходная грамматика будет:

<блок> ->

begin<список операторов>end

<список операторов> ->

<оператор> | <список операторов>; <оператор>

<оператор> ->

<идентификатор> := <выражение>

<выражение> ->

<арифметическое выражение> <операция отношения> <арифметическое выражение> |

<арифметическое выражение>

<арифметическое выражение> ->

<арифметическое выражение> <операция типа сложения> <терм> |

<терм>

<терм> ->

<терм> <операция типа умножения> <фактор> |

<фактор>

<фактор> ->

<идентификатор> |

<константа> |

(<арифметическое выражение>)

<операция отношения> ->

< | <= | = | <> | > | >=

<операция типа сложения> ->

+ | -

<операция типа умножения> ->

* | /

или ластиками буквами

<S> ->

begin<L>end

<L> ->

<O> | <L>; <O>

<O> ->

id := <X>

<X> ->

<E> <R> <E> |

<E>

<E> ->

<E> <A> <T> |

<T>

<T> ->

<T> <M> <F> |

<F>

<F> ->

id |

const |

(<E>)

<R> ->

< | <= | = | <> | > | >=

<A> ->

+ | -

<M> ->

* | /

После удаления левой рекурсивной и применения левой факторизации

S -> begin L end

L -> O B

B -> L’ | £

O -> id := X

X -> E X’

X’ -> R E | £

E -> T C

C -> E’ | £

T -> F D

D -> T’ | £

F -> id | const | (E)

R -> < | <= | = | <> | > | >=

A -> + | -

M -> * | /

L’ -> ; O B

E’ -> A T C

T’ -> M F D

Текст программы

Листинг 1 – Класс, реализующий синтаксический разбор по методу рекурсивного спуска

```
package ru.bmstu.kibamba.parsing;

import ru.bmstu.kibamba.dto.TerminalFunctionResponse;
import ru.bmstu.kibamba.models.Grammar;
import ru.bmstu.kibamba.models.GrammarSymbol;

import java.util.ArrayList;
import java.util.List;

import static ru.bmstu.kibamba.parsing.ParserUtils.*;
import static ru.bmstu.kibamba.parsing.TerminalBuilder.*;

public class Parser {
    private final Grammar grammar;
    private final List<GrammarSymbol> input;
    private int currentIndex;
    private TreeNode root;
```

```
private int erFlag;

private List<String> errorsTrace = new ArrayList<>();

public Parser(Grammar grammar, List<GrammarSymbol> input) {
    this.grammar = grammar;
    this.input = input;
}

public GrammarSymbol getCurrentInputSymbol() {
    return input.get(currentIndex);
}

private TerminalFunctionResponse S() {
    TreeNode sNode = new TreeNode(grammar.getStart());
    var a = getCurrentInputSymbol();
    if (a.equals(buildTerminalBegin())) {
        currentIndex++;
        sNode.addChild(buildTerminalNode(a));
        var l = L();
        if (l.isResult()) {
            sNode.addChild(l.getNode());
            a = getCurrentInputSymbol();
            if (a.equals(buildTerminalEnd())) {
                sNode.addChild(buildTerminalNode(a));
                return buildTerminalFunctionResponse(sNode);
            } else {
                erFlag++;
                addErrorTrace(erFlag, "end", a.getName());
                incrementFlag();
                return buildTerminalFunctionResponse();
            }
        } else {
            addErrorTrace(erFlag, "L", a.getName());
        }
    }
    incrementFlag();
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse L() {
    TreeNode lNode = buildNonterminalNode("L");
    var a = getCurrentInputSymbol();
    var o = O();
    if (o.isResult()) {
        lNode.addChild(o.getNode());
        var b = B();
        if (b.isResult()) {
            lNode.addChild(b.getNode());
            return buildTerminalFunctionResponse(lNode);
        } else {
            incrementFlag();
            addErrorTrace(erFlag, " B after O ", a.getName());
            return buildTerminalFunctionResponse();
        }
    }
    incrementFlag();
    addErrorTrace(erFlag, "O", a.getName());
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse B() {
    TreeNode bNode = buildNonterminalNode("B");
    var lPrime = LPrime();
    if (lPrime.isResult()) {
```

```
            bNode.addChild(lPrime.getNode());
            return buildTerminalFunctionResponse(bNode);
        }
        bNode.addChild(buildEpsilonNode());
        return buildTerminalFunctionResponse(bNode);
    }

    private TerminalFunctionResponse O() {
        TreeNode oNode = buildNonterminalNode("O");
        var a = getCurrentInputSymbol();
        if (a.equals(buildTerminalId())) {
            oNode.addChild(buildTerminalNode(a));
            currentIndex++;
            a = getCurrentInputSymbol();
            if (a.equals(buildTerminalIs())) {
                oNode.addChild(buildTerminalNode(a));
                currentIndex++;
                var x = X();
                if (x.isResult()) {
                    oNode.addChild(x.getNode());
                    return buildTerminalFunctionResponse(oNode);
                } else {
                    incrementFlag();
                    addErrorTrace(erFlag, "X", a.getName());
                    return buildTerminalFunctionResponse();
                }
            } else {
                incrementFlag();
                addErrorTrace(erFlag, ":=", a.getName());
                return buildTerminalFunctionResponse();
            }
        }
        incrementFlag();
        addErrorTrace(erFlag, "id", a.getName());
        return buildTerminalFunctionResponse();
    }

    private TerminalFunctionResponse LPrime() {
        TreeNode lPrimeNode = buildNonterminalNode("L'");
        var a = getCurrentInputSymbol();
        if (a.equals(buildTerminalSemicolon())) {
            lPrimeNode.addChild(buildTerminalNode(a));
            currentIndex++;
            var o = O();
            if (o.isResult()) {
                lPrimeNode.addChild(o.getNode());
                var b = B();
                if (b.isResult()) {
                    lPrimeNode.addChild(b.getNode());
                    return buildTerminalFunctionResponse(lPrimeNode);
                } else {
                    incrementFlag();
                    addErrorTrace(erFlag, "B", a.getName());
                    return buildTerminalFunctionResponse();
                }
            } else {
                incrementFlag();
                addErrorTrace(erFlag, "O", a.getName());
                return buildTerminalFunctionResponse();
            }
        }
        incrementFlag();
        addErrorTrace(erFlag, ";", a.getName());
        return buildTerminalFunctionResponse();
    }
```

```
private TerminalFunctionResponse EPrime() {
    TreeNode ePrimeNode = buildNonterminalNode("E'");
    var a = A();
    if (a.isResult()) {
        ePrimeNode.addChild(a.getNode());
        return TC(ePrimeNode);
    }
    incrementFlag();
    addErrorTrace(erFlag, "A", "Others ");
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse TC(TreeNode node) {
    var t = T();
    if (t.isResult()) {
        node.addChild(t.getNode());
        var c = C();
        if (c.isResult()) {
            node.addChild(c.getNode());
            return buildTerminalFunctionResponse(node);
        }
        incrementFlag();
        addErrorTrace(erFlag, "C", "Others ");
        return buildTerminalFunctionResponse();
    }
    incrementFlag();
    addErrorTrace(erFlag, "T", "Others ");
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse TPrime() {
    TreeNode tPrimeNode = buildNonterminalNode("T'");
    var m = M();
    if (m.isResult()) {
        tPrimeNode.addChild(m.getNode());
        return FD(tPrimeNode);
    }
    incrementFlag();
    addErrorTrace(erFlag, "M", "Others ");
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse X() {
    TreeNode xNode = buildNonterminalNode("X");
    var a = getCurrentInputSymbol();
    var e = E();
    if (e.isResult()) {
        xNode.addChild(e.getNode());
        var xPrime = XPrime();
        if (xPrime.isResult()) {
            xNode.addChild(xPrime.getNode());
            return buildTerminalFunctionResponse(xNode);
        }
        incrementFlag();
        addErrorTrace(erFlag, "X'", a.getName());

    }
    incrementFlag();
    addErrorTrace(erFlag, "E", a.getName());
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse XPrime() {
    TreeNode xPrimeNode = buildNonterminalNode("X'");
```

```
    var r = R();
    if (r.isResult()) {
        xPrimeNode.addChild(r.getNode());
        var e = E();
        if (e.isResult()) {
            xPrimeNode.addChild(e.getNode());
            return buildTerminalFunctionResponse(xPrimeNode);
        }
        incrementFlag();
        addErrorTrace(erFlag, "E", "Others ");
        return buildTerminalFunctionResponse();
    }
    xPrimeNode.addChild(buildEpsilonNode());
    return buildTerminalFunctionResponse(xPrimeNode);
}

private TerminalFunctionResponse E() {
    TreeNode eNode = buildNonterminalNode("E");
    return TC(eNode);
}

private TerminalFunctionResponse C() {
    TreeNode cNode = buildNonterminalNode("C");
    var ePrime = EPrime();
    if (ePrime.isResult()) {
        cNode.addChild(ePrime.getNode());
        return buildTerminalFunctionResponse(cNode);
    }
    cNode.addChild(buildEpsilonNode());
    return buildTerminalFunctionResponse(cNode);
}

private TerminalFunctionResponse T() {
    TreeNode tNode = buildNonterminalNode("T");
    return FD(tNode);
}

private TerminalFunctionResponse FD(TreeNode node) {
    var f = F();
    if (f.isResult()) {
        node.addChild(f.getNode());
        var d = D();
        if (d.isResult()) {
            node.addChild(d.getNode());
            return buildTerminalFunctionResponse(node);
        }
        incrementFlag();
        addErrorTrace(erFlag, "D", "Others ");
        return buildTerminalFunctionResponse();
    }
    incrementFlag();
    addErrorTrace(erFlag, "F", "Others ");
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse D() {
    TreeNode dNode = buildNonterminalNode("D");
    var tPrime = TPrime();
    if (tPrime.isResult()) {
        dNode.addChild(tPrime.getNode());
        return buildTerminalFunctionResponse(dNode);
    }
    dNode.addChild(buildEpsilonNode());
    return buildTerminalFunctionResponse(dNode);
}
```

```
private TerminalFunctionResponse F() {
    TreeNode fNode = buildNonterminalNode("F");
    var a = getCurrentInputSymbol();
    if (a.equals(buildTerminalId())) {
        currentIndex++;
        fNode.addChild(buildTerminalNode(a));
        return buildTerminalFunctionResponse(fNode);
    }

    if (a.equals(buildTerminalConst())) {
        currentIndex++;
        fNode.addChild(buildTerminalNode(a));
        return buildTerminalFunctionResponse(fNode);
    }

    if (a.equals(buildTerminalLParen())) {
        fNode.addChild(buildTerminalNode(a));
        currentIndex++;
        var e = E();
        if (e.isResult()) {
            fNode.addChild(e.getNode());
            a = getCurrentInputSymbol();
            if (a.equals(buildTerminalRParen())) {
                currentIndex++;
                fNode.addChild(buildTerminalNode(a));
                return buildTerminalFunctionResponse(fNode);
            }
            incrementFlag();
            addErrorTrace(erFlag, ")", a.getName());
            return buildTerminalFunctionResponse();
        }
        incrementFlag();
        addErrorTrace(erFlag, "E", "Others ");
        return buildTerminalFunctionResponse();
    }
    incrementFlag();
    addErrorTrace(erFlag, "id , const or (E)", "Others ");
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse M() {
    var a = getCurrentInputSymbol();
    TreeNode mNode = buildNonterminalNode("M");
    if (a.equals(buildTerminalMul())) {
        currentIndex++;
        mNode.addChild(buildTerminalNode(a));
        return buildTerminalFunctionResponse(mNode);
    } else if (a.equals(buildTerminalDiv())) {
        currentIndex++;
        mNode.addChild(buildTerminalNode(a));
        return buildTerminalFunctionResponse(mNode);
    }
    incrementFlag();
    addErrorTrace(erFlag, "* or /", a.getName());
    return buildTerminalFunctionResponse();
}

private TerminalFunctionResponse A() {
    var a = getCurrentInputSymbol();
    TreeNode aNode = buildNonterminalNode("A");
    if (a.equals(buildTerminalAdd())) {
        currentIndex++;
        aNode.addChild(buildTerminalNode(a));
        return buildTerminalFunctionResponse(aNode);
```

```
        } else if (a.equals(buildTerminalSub())) {
            currentIndex++;
            aNode.addChild(buildTerminalNode(a));
            return buildTerminalFunctionResponse(aNode);
        }
        incrementFlag();
        addErrorTrace(erFlag, "+ or -", a.getName());
        return buildTerminalFunctionResponse();
    }

    private TerminalFunctionResponse R() {
        var a = getCurrentInputSymbol();
        TreeNode rNode = buildNonterminalNode("R");

        if (a.equals(buildTerminalLess())) {
            currentIndex++;
            rNode.addChild(buildTerminalNode(a));
            return buildTerminalFunctionResponse(rNode);
        } else if (a.equals(buildTerminalLessEqual())) {
            currentIndex++;
            rNode.addChild(buildTerminalNode(a));
            return buildTerminalFunctionResponse(rNode);
        } else if (a.equals(buildTerminalEqual())) {
            currentIndex++;
            rNode.addChild(buildTerminalNode(a));
            return buildTerminalFunctionResponse(rNode);
        } else if (a.equals(buildTerminalNotEqual())) {
            currentIndex++;
            rNode.addChild(buildTerminalNode(a));
            return buildTerminalFunctionResponse(rNode);
        } else if (a.equals(buildTerminalGreat())) {
            currentIndex++;
            rNode.addChild(buildTerminalNode(a));
            return buildTerminalFunctionResponse(rNode);
        } else if (a.equals(buildTerminalGreatEqual())) {
            currentIndex++;
            rNode.addChild(buildTerminalNode(a));
            return buildTerminalFunctionResponse(rNode);
        }
        incrementFlag();
        addErrorTrace(erFlag, "<, >, <=, >=, =,<>", a.getName());
        return buildTerminalFunctionResponse();
    }

    private void addErrorTrace(int errorNumber, String expected, String found) {
        errorsTrace.add("ERROR " + errorNumber + " expected " + expected + " but
found " + found + "\n");
    }

    private void printErrorTrace() {
        for (String error : errorsTrace) {
            System.out.print(error);
        }
    }

    private void incrementFlag() {
        erFlag++;
    }

    public boolean parse() {
        currentIndex = 0;
        erFlag = 0;
        TerminalFunctionResponse response = S();
        root = response.getNode();
        if (response.isResult()) {
```

```
            return true;
        } else {
            if (erFlag > 0) {
                System.out.println("INTERNAL ERROR");
                System.out.println("Not expected \'"+getCurrentInputSymbol().get-
Name()+"\' at "+currentIndex+" position ");
            } else {
                System.out.println("Position " + currentIndex);
                System.out.println("Error: Incorrect  first symbol of S!");
            }
            /*System.out.println("\nTrace");*/
            //printErrorTrace();
        }
        return false;
    }

    public TreeNode getRoot() {
        return root;
    }
}
```

Листинг 2 – Класс дерева

```
package ru.bmstu.kibamba.parsing;

import ru.bmstu.kibamba.models.GrammarSymbol;

import java.util.ArrayList;
import java.util.List;

public class TreeNode {
    GrammarSymbol value;
    List<TreeNode> children;

    TreeNode(GrammarSymbol value) {
        this.value = value;
        this.children = new ArrayList<>();
    }

    void addChild(TreeNode child) {
        children.add(child);
    }

    @Override
    public String toString() {
        return toString("", true, true);
    }

    private String toString(String indent, boolean isLast, boolean isRoot) {
        StringBuilder sb = new StringBuilder();
        sb.append(isRoot ? "" : indent).append(isRoot ? "" : "|--").append(value.get-
Name()).append("\n");
        for (int i = 0; i < children.size(); i++) {
            sb.append(children.get(i).toString(indent + (isLast ? isRoot ? "" : "   "
: "|  "),
                    i == children.size() - 1, false));
        }
        return sb.toString();
    }
}
```

Листинг 3 – Основной класс запуски программы

```
import ru.bmstu.kibamba.files.TerminalFileReader;
import ru.bmstu.kibamba.models.*;
import ru.bmstu.kibamba.parsing.Parser;
```

```
import ru.bmstu.kibamba.parsing.ParserUtils;

import java.util.*;

import static ru.bmstu.kibamba.parsing.TerminalBuilder.*;

public class Main {
    public static void main(String[] args) {
        Set<Nonterminal> nonterminals = new LinkedHashSet<>();
        Set<Production> productions = new LinkedHashSet<>();
        Set<Terminal> terminals = new LinkedHashSet<>();

        var nonTerminalS = new Nonterminal("S", true);
        var nonTerminalL = new Nonterminal("L");
        var nonTerminalB = new Nonterminal("B");
        var nonTerminalO = new Nonterminal("O");
        var nonTerminalX = new Nonterminal("X");
        var nonTerminalXPrime = new Nonterminal("X'");
        var nonTerminalE = new Nonterminal("E");
        var nonTerminalC = new Nonterminal("C");
        var nonTerminalT = new Nonterminal("T");
        var nonTerminalD = new Nonterminal("D");
        var nonTerminalF = new Nonterminal("F");
        var nonTerminalR = new Nonterminal("R");
        var nonTerminalA = new Nonterminal("A");
        var nonTerminalM = new Nonterminal("M");
        var nonTerminalLPrime = new Nonterminal("L'");
        var nonTerminalEPrime = new Nonterminal("E'");
        var nonTerminalTPrime = new Nonterminal("T'");


        Collections.addAll(nonterminals, nonTerminalS, nonTerminalL, nonTerminalB,
                nonTerminalO, nonTerminalX, nonTerminalXPrime, nonTerminalE, nonTer-
minalC,
                nonTerminalT, nonTerminalD, nonTerminalF, nonTerminalR, nonTerminalA,
nonTerminalM,
                nonTerminalLPrime, nonTerminalEPrime, nonTerminalTPrime);

        var terminalBegin = buildTerminalBegin();
        var terminalEnd = buildTerminalEnd();
        var terminalVar = buildTerminalId();
        var terminalIs = buildTerminalIs();
        var terminalSemicolon = buildTerminalSemicolon();
        var terminalConst = buildTerminalConst();
        var terminalLParen = buildTerminalLParen();
        var terminalRParen = buildTerminalRParen();
        var terminalMul = buildTerminalMul();
        var terminalDiv = buildTerminalDiv();
        var terminalAdd = buildTerminalAdd();
        var terminalSub = buildTerminalSub();
        var terminalLess = buildTerminalLess();
        var terminalLessEqual = buildTerminalLessEqual();
        var terminalEqual = buildTerminalEqual();
        var terminalNotEqual = buildTerminalNotEqual();
        var terminalGreat = buildTerminalGreat();
        var terminalGreatEqual = buildTerminalGreatEqual();

        var epsilon = new GrammarSymbol("£");

        Collections.addAll(terminals, terminalBegin, terminalEnd,
                terminalVar, terminalIs, terminalSemicolon, terminalConst,
                terminalLParen, terminalRParen, terminalMul, terminalDiv, terminal-
Add,
                terminalLess, terminalEqual, terminalLessEqual, terminalNotEqual,
                terminalGreat, terminalGreatEqual, terminalSub);
```

```
        var productionS = ParserUtils.buildProduction(nonTerminalS, terminalBegin,
nonTerminalL, terminalEnd);

        var productionL = ParserUtils.buildProduction(nonTerminalL, nonTerminalO,
nonTerminalB);
        var productionB = ParserUtils.buildProduction(nonTerminalB, nonTerminalL-
Prime);
        var productionBSc = ParserUtils.buildProduction(nonTerminalB, epsilon);
        var productionO = ParserUtils.buildProduction(nonTerminalO, terminalVar, ter-
minalIs, nonTerminalX);
        var productionX = ParserUtils.buildProduction(nonTerminalX, nonTerminalE,
nonTerminalXPrime);
        var productionXPrime = ParserUtils.buildProduction(nonTerminalXPrime, nonTer-
minalR, nonTerminalE);
        var productionXPrimeSc = ParserUtils.buildProduction(nonTerminalXPrime, epsi-
lon);
        var productionE = ParserUtils.buildProduction(nonTerminalE, nonTerminalT,
nonTerminalC);
        var productionC = ParserUtils.buildProduction(nonTerminalC, nonTermi-
nalEPrime);
        var productionCSc = ParserUtils.buildProduction(nonTerminalC, epsilon);
        var productionT = ParserUtils.buildProduction(nonTerminalT, nonTerminalF,
nonTerminalD);
        var productionD = ParserUtils.buildProduction(nonTerminalD, nonTermi-
nalTPrime);
        var productionDSc = ParserUtils.buildProduction(nonTerminalD, epsilon);
        var productionF = ParserUtils.buildProduction(nonTerminalF, terminalVar);
        var productionFSc = ParserUtils.buildProduction(nonTerminalF, terminalConst);
        var productionFTh = ParserUtils.buildProduction(nonTerminalF, terminalLParen,
nonTerminalE, terminalRParen);
        var productionRL = ParserUtils.buildProduction(nonTerminalR, terminalLess);
        var productionRLE = ParserUtils.buildProduction(nonTerminalR, terminalLessE-
qual);
        var productionRE = ParserUtils.buildProduction(nonTerminalR, terminalEqual);
        var productionLPrime = ParserUtils.buildProduction(nonTerminalLPrime, termi-
nalSemicolon, nonTerminalO, nonTerminalB);
        var productionEPrime = ParserUtils.buildProduction(nonTerminalEPrime, nonTer-
minalA, nonTerminalT, nonTerminalC);
        var productionTPrime = ParserUtils.buildProduction(nonTerminalTPrime, nonTer-
minalM, nonTerminalF, nonTerminalD);
        var productionRNE = ParserUtils.buildProduction(nonTerminalR, terminalNotE-
qual);
        var productionRG = ParserUtils.buildProduction(nonTerminalR, terminalGreat);
        var productionRGE = ParserUtils.buildProduction(nonTerminalR, terminalGreatE-
qual);
        var productionAAdd = ParserUtils.buildProduction(nonTerminalA, terminalAdd);
        var productionASub = ParserUtils.buildProduction(nonTerminalA, terminalSub);
        var productionMMul = ParserUtils.buildProduction(nonTerminalM, terminalMul);
        var productionMDiv = ParserUtils.buildProduction(nonTerminalM, terminalDiv);

        Collections.addAll(productions, productionS, productionL, productionB, pro-
ductionBSc, productionO,
                productionX, productionXPrime, productionXPrimeSc, productionE, pro-
ductionC, productionCSc,
                productionT, productionD, productionDSc, productionF, productionFSc,
productionFTh,
                productionRL, productionRLE, productionRE, productionRNE, produc-
tionRG, productionRGE, productionAAdd,
                productionASub, productionMMul, productionMDiv, productionLPrime,
productionEPrime, productionTPrime);
```

```
        Grammar grammar = new Grammar(nonterminals, terminals, nonTerminalS, productions);

        List<GrammarSymbol> inputExample01 = TerminalFileReader.buildInputChain("example01");

        List<GrammarSymbol> inputExample02 = TerminalFileReader.buildInputChain("example02");

        List<GrammarSymbol> inputWithError = TerminalFileReader.buildInputChain("example_with_error");

        Parser parser = new Parser(grammar, inputExample01);
        var isParsed = parser.parse();
        if (isParsed) {
            System.out.println(parser.getRoot());
        }


    }
}
```

Набор тестов и ожидаемые результаты для проверки правильности программы

1. Пример 1.

Входной текстовой файл:

```
begin
    a := 5
end
```

Результат:

S

|--begin

|--L

| |--O

| | |--id

| | |--:=

| | |--X

| | |--E

| | | |--T

| | | | |--F

| | | | | |--const

| | | |--D

| | | |--£

| | |--C

| | | |--£

| | |--X'

| | |--£

| |--B

| |--£

|--end

2. Пример 2

Входной файл

```
begin
    b := 2 ;
    c := 3 ;
    a := b * 2 + ( c - 7 ) <> b - 7
end
```

Результат

S

|--begin

|--L

| |--O

| | |--id

| | |--:=

| | |--X

| | |--E

| | | |--T

| | | | |--F

| | | | | |--const

| | | | |--D

| | | | |--£

| | | |--C

| | | |--£

| | |--X'

| | |--£

| |--B

| |--L'

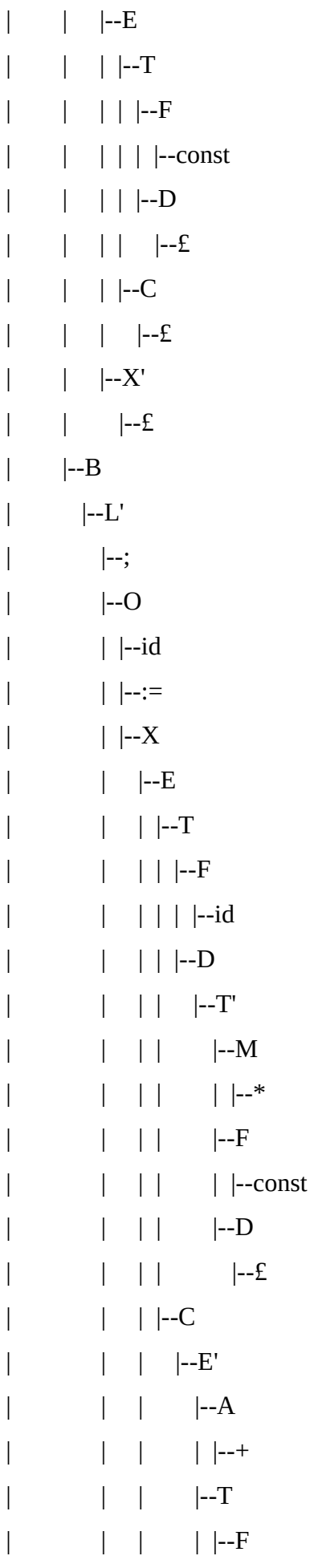
| |--;

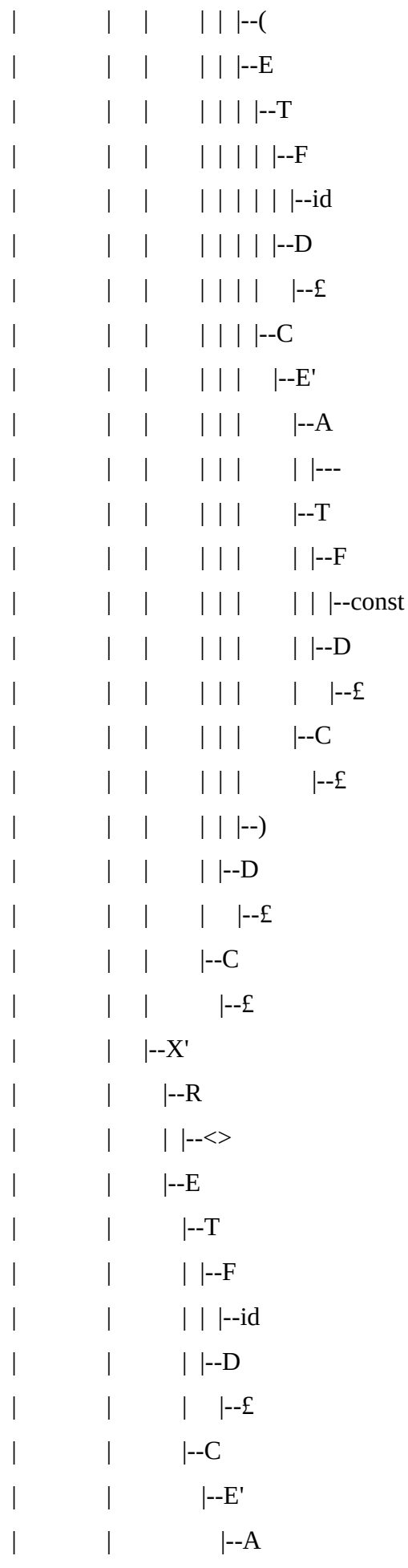
| |--O

| | |--id

| | |--:=

| | |--X


|--E
|--T
|--F
|--const
|--D
|--£
|--C
|--£
|--X'
|--£
|--B
|--L'
|--;
|--O
|--id
|--:=
|--X
|--E
|--T
|--F
|--id
|--D
|--T'
|--M
|--*
|--F
|--const
|--D
|--£
|--C
|--E'
|--A
|--+
|--T
|--F


|--(
|--E
|--T
|--F
|--id
|--D
|--£
|--C
|--E'
|--A
|---
|--T
|--F
|--const
|--D
|--£
|--C
|--£
|--)
|--D
|--£
|--C
|--£
|--X'
|--R
|--<>
|--E
|--T
|--F
|--id
|--D
|--£
|--C
|--E'
|--A

| | | |---
| | |--T
| | | |--F
| | | | |--const
| | | |--D
| | | |--£
| | |--C
| | |--£
| |--B
| |--£
|--end

3. Пример 3

Входной файл

```
begin
    b := 2 ;
    c := 3 ;
    a := b * 2 +  c - 7 ) <> b - 7
end
```

Результат

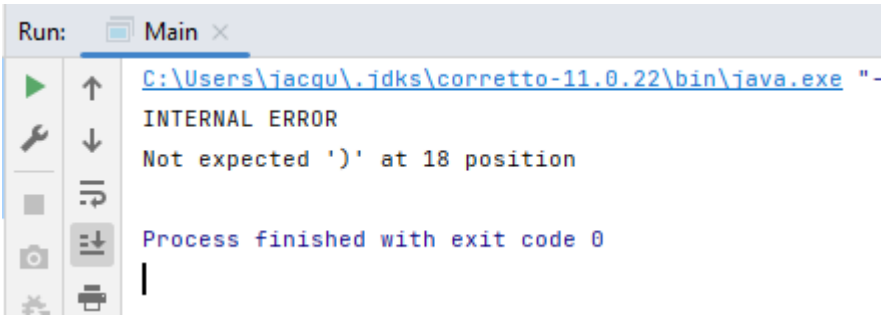


Рис. 1 - Результат примера с ошибкой в коде