

# Construção de Compiladores 1 - 2018.1 - Profs. Mário César San Felice (e Helena Caseli, Murilo Naldi, Daniel Lucrédio)

## Tópico 04 - Análise Sintática Descendente - Roteiro de Códigos

### Demonstração 1 – Analisador sintático preditivo de descendência recursiva “na mão”

---

1. Criar um novo arquivo, no Desktop, com um programa de exemplo

```
:DECLARACOES
numero1:INTEIRO
numero2:INTEIRO
numero3:INTEIRO
aux:INTEIRO

:ALGORITMO
% Coloca 3 números em ordem crescente
LER numero1
LER numero2
LER numero3
SE numero1 > numero2 ENTÃO
  INICIO
    ATRIBUIR 2+3-4+5-6*5-1 A aux
    ATRIBUIR numero1 A numero2
    ATRIBUIR aux A numero1
  FIM
SE numero1 > numero3 E numero2 <= numero4 E numero1 > 3 OU numero2 <> numero4
ENTÃO
  INICIO
    ATRIBUIR (numero3) A aux
    ATRIBUIR numero1 A numero3
    ATRIBUIR aux A numero1
  FIM
SE numero2 > numero3 ENTÃO
  INICIO
    ATRIBUIR numero3 A aux
    ATRIBUIR numero2 A numero3
    ATRIBUIR aux A numero2
  FIM
IMPRIMIR numero1
IMPRIMIR numero2
IMPRIMIR numero3
```

2. Abrir o NetBeans, e abrir projeto Java “AlgumaLex”

3. Criar novo projeto Java “AlgumaParser”

4. Adicionar dependência do “AlgumaParser” para o “AlgumaLex”

5. Criar um arquivo com a gramática

```
programa : ':' 'DECLARACOES' listaDeclaracoes ':' 'ALGORITMO' listaComandos;
listaDeclaracoes : declaracao listaDeclaracoes | declaracao;
declaracao : VARIAVEL ':' tipoVar;
```

```

tipoVar : 'INTEIRO' | 'REAL';
expressaoAritmetica : expressaoAritmetica '+' termoAritmetico |
expressaoAritmetica '-' termoAritmetico | termoAritmetico;
termoAritmetico : termoAritmetico '*' fatorAritmetico | termoAritmetico '/'
fatorAritmetico | fatorAritmetico;
fatorAritmetico : NUMINT | NUMREAL | VARIAVEL | '(' expressaoAritmetica ')'
expressaoRelacional : expressaoRelacional operadorBooleano termoRelacional |
termoRelacional;
termoRelacional : expressaoAritmetica OP_REL expressaoAritmetica | '('
expressaoRelacional ')';
operadorBooleano : 'E' | 'OU';
listaComandos : comando listaComandos | comando;
comando : comandoAtribuicao | comandoEntrada | comandoSaida | comandoCondicao
| comandoRepeticao | subAlgoritmo;
comandoAtribuicao : 'ATRIBUIR' expressaoAritmetica 'A' VARIAVEL;
comandoEntrada : 'LER' VARIAVEL;
comandoSaida : 'IMPRIMIR' (VARIAVEL | CADEIA);
comandoCondicao : 'SE' expressaoRelacional 'ENTAO' comando | 'SE'
expressaoRelacional 'ENTAO' comando 'SENAO' comando;
comandoRepeticao : 'ENQUANTO' expressaoRelacional comando;
subAlgoritmo : 'INICIO' listaComandos 'FIM';

```

## 6. Criar uma classe algumaparser.AlgumaParser

```

package algumaparser;

import algumalex.AlgumaLexico;
import algumalex.TipoToken;
import algumalex.Token;
import java.util.ArrayList;
import java.util.List;

public class AlgumaParser {

    private final static int TAMANHO_BUFFER = 10;
    List<Token> bufferTokens;
    AlgumaLexico lex;
    boolean chegouNoFim = false;

    public AlgumaParser(AlgumaLexico lex) {
        this.lex = lex;
        bufferTokens = new ArrayList<Token>();
        lerToken();
    }

    private void lerToken() {
        if (bufferTokens.size() > 0) {
            bufferTokens.remove(0);
        }
        while (bufferTokens.size() < TAMANHO_BUFFER && !chegouNoFim) {
            Token proximo = lex.proximoToken();

```

```

        bufferTokens.add(proximo);
        if (proximo.nome == TipoToken.Fim) {
            chegouNoFim = true;
        }
    }
    System.out.println("Lido: " + lookahead(1));
}

void match(TipoToken tipo) {
    if (lookahead(1).nome == tipo) {
        System.out.println("Match: " + lookahead(1));
        lerToken();
    } else {
        erroSintatico(tipo.toString());
    }
}

void match(TipoToken tipo, String lexema) {
    if (lookahead(1).nome == tipo && lookahead(1).lexema.equals(lexema))
    {
        System.out.println("Match: " + lookahead(1));
        lerToken();
    } else {
        erroSintatico(lexema);
    }
}

Token lookahead(int k) {
    if (bufferTokens.isEmpty()) {
        return null;
    }
    if (k - 1 >= bufferTokens.size()) {
        return bufferTokens.get(bufferTokens.size() - 1);
    }
    return bufferTokens.get(k - 1);
}

void erroSintatico(String... tokensEsperados) {
    String mensagem = "Erro sintático: esperando um dos seguintes (";
    for(int i=0;i<tokensEsperados.length;i++) {
        mensagem += tokensEsperados[i];
        if(i<tokensEsperados.length-1)
            mensagem += ",";
    }
    mensagem += "), mas foi encontrado " + lookahead(1);
    throw new RuntimeException(mensagem);
}

```

7. Copiar a gramática para dentro do arquivo AlgumaParser.java, comentando suas linhas
8. Transformar cada linha em um método

```

//programa : ':' 'DECLARACOES' listaDeclaracoes ':' 'ALGORITMO'
listaComandos;
public void programa() {
    match(TipoToken.Delim);
    match(TipoToken.PalavraChave, "DECLARACOES");
    listaDeclaracoes();
    match(TipoToken.Delim);
    match(TipoToken.PalavraChave, "ALGORITMO");
    listaComandos();
    match(TipoToken.Fim);
}

//listaDeclaracoes : declaracao listaDeclaracoes | declaracao;
void listaDeclaracoes() {
    // usaremos lookahead(4)
    // Mas daria para fatorar à esquerda
    // Veja na lista de comandos um exemplo
    if (lookahead(4).nome == TipoToken.Delim) {
        declaracao();
    } else if (lookahead(4).nome == TipoToken.Var) {
        declaracao();
        listaDeclaracoes();
    } else {
        erroSintatico(TipoToken.Delim.toString(),
TipoToken.Var.toString());
    }
}

//declaracao : VARIAVEL ':' tipoVar;
void declaracao() {
    match(TipoToken.Var);
    match(TipoToken.Delim);
    tipoVar();
}

//tipoVar : 'INTEIRO' | 'REAL';
void tipoVar() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("INTEIRO")) {
        match(TipoToken.PalavraChave, "INTEIRO");
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("REAL")) {
        match(TipoToken.PalavraChave, "REAL");
    } else {
        erroSintatico("INTEIRO", "REAL");
    }
}

//expressaoAritmetica : expressaoAritmetica '+' termoAritmetico |
expressaoAritmetica '-' termoAritmetico | termoAritmetico;
// fatorar à esquerda:

```

```

// expressaoAritmetica : expressaoAritmetica ('+' termoAritmetico | '-'
termoAritmetico) | termoAritmetico;
// fatorar não é suficiente, pois causa loop infinito
// remover a recursão à esquerda
// expressaoAritmetica : termoAritmetico expressaoAritmetica2
// expressaoAritmetica2 : ('+' termoAritmetico | '-' termoAritmetico)
expressaoAritmetica2 | <<vazio>>
void expressaoAritmetica() {
    termoAritmetico();
    expressaoAritmetica2();
}

void expressaoAritmetica2() {
    if (lookahead(1).nome == TipoToken.OpAritSoma || lookahead(1).nome ==
TipoToken.OpAritSub) {
        expressaoAritmetica2SubRegral();
        expressaoAritmetica2();
    } else { // vazio
    }
}

void expressaoAritmetica2SubRegral() {
    if (lookahead(1).nome == TipoToken.OpAritSoma) {
        match(TipoToken.OpAritSoma);
        termoAritmetico();
    } else if (lookahead(1).nome == TipoToken.OpAritSub) {
        match(TipoToken.OpAritSub);
        termoAritmetico();
    } else {
        erroSintatico("+", "-");
    }
}

//termoAritmetico : termoAritmetico '*' fatorAritmetico | termoAritmetico
 '/' fatorAritmetico | fatorAritmetico;
// também precisa fatorar à esquerda e eliminar recursão à esquerda
// termoAritmetico : fatorAritmetico termoAritmetico2
// termoAritmetico2 : ('*' fatorAritmetico | '/' fatorAritmetico)
termoAritmetico2 | <<vazio>>
void termoAritmetico() {
    fatorAritmetico();
    termoAritmetico2();
}

void termoAritmetico2() {
    if (lookahead(1).nome == TipoToken.OpAritMult || lookahead(1).nome ==
TipoToken.OpAritDiv) {
        termoAritmetico2SubRegral();
        termoAritmetico2();
    } else { // vazio
    }
}

```

```

}

void termoAritmetico2SubRegal() {
    if (lookahead(1).nome == TipoToken.OpAritMult) {
        match(TipoToken.OpAritMult);
        fatorAritmetico();
    } else if (lookahead(1).nome == TipoToken.OpAritDiv) {
        match(TipoToken.OpAritDiv);
        fatorAritmetico();
    } else {
        erroSintatico("*", "/");
    }
}

//fatorAritmetico : NUMINT | NUMREAL | VARIAVEL | '(' expressaoAritmetica
')'
void fatorAritmetico() {
    if (lookahead(1).nome == TipoToken.NumInt) {
        match(TipoToken.NumInt);
    } else if (lookahead(1).nome == TipoToken.NumReal) {
        match(TipoToken.NumReal);
    } else if (lookahead(1).nome == TipoToken.Var) {
        match(TipoToken.Var);
    } else if (lookahead(1).nome == TipoToken.AbrePar) {
        match(TipoToken.AbrePar);
        expressaoAritmetica();
        match(TipoToken.FechaPar);
    } else {

erroSintatico(TipoToken.NumInt.toString(), TipoToken.NumReal.toString(), TipoTo
ken.Var.toString(), "(");
    }
}

//expressaoRelacional : expressaoRelacional operadorBooleano
termoRelacional | termoRelacional;
// Precisa eliminar a recursão à esquerda
// expressaoRelacional : termoRelacional expressaoRelacional2;
// expressaoRelacional2 : operadorBooleano termoRelacional
expressaoRelacional2 | <<vazio>>
void expressaoRelacional() {
    termoRelacional();
    expressaoRelacional2();
}

void expressaoRelacional2() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
(lookahead(1).lexema.equals("E") || lookahead(1).lexema.equals("OU"))) {
        operadorBooleano();
        termoRelacional();
        expressaoRelacional2();
    }
}

```

```

    } else { // vazio
    }
}

//termoRelacional : expressaoAritmetica OP_REL expressaoAritmetica | '('
expressaoRelacional ')';
void termoRelacional() {
    if (lookahead(1).nome == TipoToken.NumInt
        || lookahead(1).nome == TipoToken.NumReal
        || lookahead(1).nome == TipoToken.Var
        || lookahead(1).nome == TipoToken.AbrePar) {
        // Há um não-determinismo aqui.
        // AbrePar pode ocorrer tanto em expressaoAritmetica como em
(expressaoRelacional)
        // Tem uma forma de resolver este problema, mas não usaremos aqui
        // Vamos modificar a linguagem, eliminando a possibilidade
        // de agrupar expressões relacionais com parêntesis
        expressaoAritmetica();
        opRel();
        expressaoAritmetica();
    } else {

erroSintatico(TipoToken.NumInt.toString(), TipoToken.NumReal.toString(), TipoTo
ken.Var.toString(), "");
    }
}

void opRel() {
    if (lookahead(1).nome == TipoToken.OpRelDif) {
        match(TipoToken.OpRelDif);
    } else if (lookahead(1).nome == TipoToken.OpRelIgual) {
        match(TipoToken.OpRelIgual);
    } else if (lookahead(1).nome == TipoToken.OpRelMaior) {
        match(TipoToken.OpRelMaior);
    } else if (lookahead(1).nome == TipoToken.OpRelMaiorIgual) {
        match(TipoToken.OpRelMaiorIgual);
    } else if (lookahead(1).nome == TipoToken.OpRelMenor) {
        match(TipoToken.OpRelMenor);
    } else if (lookahead(1).nome == TipoToken.OpRelMenorIgual) {
        match(TipoToken.OpRelMenorIgual);
    } else {
        erroSintatico("<>", "=", ">", ">=", "<", "<=");
    }
}

//operadorBooleano : 'E' | 'OU';
void operadorBooleano() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("E")) {
        match(TipoToken.PalavraChave, "E");
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&

```

```

lookahead(1).lexema.equals("OU")) {
    match(TipoToken.PalavraChave, "OU");
} else {
    erroSintatico("E", "OU");
}
}

//listaComandos : comando listaComandos | comando;
// vamos fatorar à esquerda
// listaComandos : comando (listaComandos | <<vazio>>)
void listaComandos() {
    comando();
    listaComandosSubRegral();
}

void listaComandosSubRegral() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
(lookahead(1).lexema.equals("ATRIBUIR")
        || lookahead(1).lexema.equals("LER")
        || lookahead(1).lexema.equals("IMPRIMIR")
        || lookahead(1).lexema.equals("SE")
        || lookahead(1).lexema.equals("ENQUANTO")
        || lookahead(1).lexema.equals("INICIO"))) {
        listaComandos();
    } else {
        // vazio
    }
}

//comando : comandoAtribuicao | comandoEntrada | comandoSaida |
comandoCondicao | comandoRepeticao | subAlgoritmo;
void comando() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("ATRIBUIR")) {
        comandoAtribuicao();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("LER")) {
        comandoEntrada();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("IMPRIMIR")) {
        comandoSaida();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("SE")) {
        comandoCondicao();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("ENQUANTO")) {
        comandoRepeticao();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("INICIO")) {
        subAlgoritmo();
    } else {

```



```

erroSintatico("ATRIBUIR","LER","IMPRIMIR","SE","ENQUANTO","INICIO");
    }
}

//comandoAtribuicao : 'ATRIBUIR' expressaoAritmetica 'A' VARIAVEL;
void comandoAtribuicao() {
    match(TipoToken.PalavraChave, "ATRIBUIR");
    expressaoAritmetica();
    match(TipoToken.PalavraChave, "A");
    match(TipoToken.Var);
}

//comandoEntrada : 'LER' VARIAVEL;
void comandoEntrada() {
    match(TipoToken.PalavraChave, "LER");
    match(TipoToken.Var);
}

//comandoSaida : 'IMPRIMIR' (VARIAVEL | CADEIA);
void comandoSaida() {
    match(TipoToken.PalavraChave, "IMPRIMIR");
    comandoSaidaSubRegal();
}

void comandoSaidaSubRegal() {
    if (lookahead(1).nome == TipoToken.Var) {
        match(TipoToken.Var);
    } else if (lookahead(1).nome == TipoToken.Cadeia) {
        match(TipoToken.Cadeia);
    } else {

erroSintatico(TipoToken.Var.toString(),TipoToken.Cadeia.toString());
    }
}

//comandoCondicao : 'SE' expressaoRelacional 'ENTAO' comando | 'SE'
expressaoRelacional 'ENTAO' comando 'SENAO' comando;
// fatorar à esquerda
// comandoCondicao : 'SE' expressaoRelacional 'ENTAO' comando ('SENAO'
comando | <<vazio>>)
void comandoCondicao() {
    match(TipoToken.PalavraChave, "SE");
    expressaoRelacional();
    match(TipoToken.PalavraChave, "ENTAO");
    comando();
    comandoCondicaoSubRegal();
}

void comandoCondicaoSubRegal() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&

```

```

lookahead(1).lexema.equals("SENAO")) {
    match(TipoToken.PalavraChave, "SENAO");
    comando();
} else {
    // vazio
}
}

//comandoRepeticao : 'ENQUANTO' expressaoRelacional comando;
void comandoRepeticao() {
    match(TipoToken.PalavraChave, "ENQUANTO");
    expressaoRelacional();
    comando();
}

//subAlgoritmo : 'INICIO' listaComandos 'FIM';
void subAlgoritmo() {
    match(TipoToken.PalavraChave, "INICIO");
    listaComandos();
    match(TipoToken.PalavraChave, "FIM");
}

```

## 9. Criar a classe algumaparser.Principal

```

public class Principal {
    public static void main(String args[]) {
        AlgumaLexico lex = new AlgumaLexico("/home/daniel/programa.alg");
        AlgumaParser parser = new AlgumaParser(lex);
        parser.programa();
    }
}

```

### 9.1. Testar

## 10. Modificar para gerar uma árvore de sintaxe abstrata

### 10.1. Criar o elemento básico da árvore (class algumaparser.AsaNo)

```

package algumaparser;

import algumalex.Token;
import java.util.LinkedList;

public class AsaNo {
    String nome;
    Token token;
    LinkedList<AsaNo> filhos;

    public AsaNo(String nome) {
        this.nome = nome;
        filhos = new LinkedList<AsaNo>();
    }
}

```

```

public void adicionarFilho(AsaNo filho) {
    filhos.add(filho);
}

public void adicionarFilhoEsquerda(AsaNo filho) {
    filhos.addFirst(filho);
}

@Override
public String toString() {
    return toString(0);
}

protected String toString(int ident) {
    String ret = "\n"+ident(ident)+"# "+nome;
    if(token != null)
        ret += " "+token.toString();
    for(AsaNo filho:filhos) {
        ret += filho.toString(ident + 3);
    }
    return ret;
}

protected String ident(int ident) {
    String ret = "";
    for(int i=0;i<ident;i++) {
        ret += " ";
    }
    return ret;
}
}
}

```

## 10.2. Modificar as regras para construir a árvore

```

//programa : ':' 'DECLARACOES' listaDeclaracoes ':' 'ALGORITMO'
listaComandos;
public AsaNo programa() {
    AsaNo no = new AsaNo("programa");
    match(TipoToken.Delim);
    match(TipoToken.PalavraChave, "DECLARACOES");
    AsaNo listaDecs = new AsaNo("variaveis");
    listaDeclaracoes(listaDecs);
    match(TipoToken.Delim);
    match(TipoToken.PalavraChave, "ALGORITMO");
    AsaNo listaCmds = new AsaNo("algoritmo");
    listaComandos(listaCmds);
    match(TipoToken.Fim);
    no.adicionarFilho(listaDecs);
    no.adicionarFilho(listaCmds);
    return no;
}

```

```

//listaDeclaracoes : declaracao listaDeclaracoes | declaracao;
void listaDeclaracoes(AsaNo variaveis) {
    // usaremos lookahead(4)
    // Mas daria para fatorar à esquerda
    // Veja na lista de comandos um exemplo
    if (lookahead(4).nome == TipoToken.Delim) {
        AsaNo var = declaracao();
        variaveis.adicionarFilho(var);
    } else if (lookahead(4).nome == TipoToken.Var) {
        AsaNo var = declaracao();
        variaveis.adicionarFilho(var);
        listaDeclaracoes(variaveis);
    } else {
        erroSintatico(TipoToken.Delim.toString(),
TipoToken.Var.toString());
    }
}

//declaracao : VARIAVEL ':' tipoVar;
AsaNo declaracao() {
    AsaNo ret = new AsaNo("");
    Token varToken = lookahead(1);
    match(TipoToken.Var);
    match(TipoToken.Delim);
    String tipo = tipoVar();
    ret.nome = varToken.lexema + ":" + tipo;
    ret.token = varToken;
    return ret;
}

//tipoVar : 'INTEIRO' | 'REAL';
String tipoVar() {
    String ret = null;
    if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("INTEIRO")) {
        ret = "INTEIRO";
        match(TipoToken.PalavraChave, "INTEIRO");
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("REAL")) {
        ret = "REAL";
        match(TipoToken.PalavraChave, "REAL");
    } else {
        erroSintatico("INTEIRO", "REAL");
    }
    return ret;
}

//expressaoAritmetica : expressaoAritmetica '+' termoAritmetico |
expressaoAritmetica '-' termoAritmetico | termoAritmetico;
// fatorar à esquerda:

```

```

// expressaoAritmetica : expressaoAritmetica ('+' termoAritmetico | '-'
termoAritmetico) | termoAritmetico;
// fatorar não é suficiente, pois causa loop infinito
// remover a recursão à esquerda
// expressaoAritmetica : termoAritmetico expressaoAritmetica2
// expressaoAritmetica2 : ('+' termoAritmetico | '-' termoAritmetico)
expressaoAritmetica2 | <<vazio>>
AsaNo expressaoAritmetica() {
    AsaNo termo = termoAritmetico();
    AsaNo termo2 = expressaoAritmetica2(termo);
    if (termo2 != null) {
        return termo2;
    }
    return termo;
}

AsaNo expressaoAritmetica2(AsaNo termol) {
    if (lookahead(1).nome == TipoToken.OpAritSoma || lookahead(1).nome ==
TipoToken.OpAritSub) {
        AsaNo expr1 = expressaoAritmetica2SubRegral();
        expr1.adicionarFilhoEsquerda(termol);
        AsaNo expr2 = expressaoAritmetica2(expr1);
        if (expr2 != null) {
            return expr2;
        }
        return expr1;
    } else { // vazio
        return null;
    }
}

AsaNo expressaoAritmetica2SubRegral() {
    if (lookahead(1).nome == TipoToken.OpAritSoma) {
        AsaNo ret = new AsaNo("+");
        ret.token = lookahead(1);
        match(TipoToken.OpAritSoma);
        AsaNo termo = termoAritmetico();
        ret.adicionarFilho(termo);
        return ret;
    } else if (lookahead(1).nome == TipoToken.OpAritSub) {
        AsaNo ret = new AsaNo("-");
        ret.token = lookahead(1);
        match(TipoToken.OpAritSub);
        AsaNo termo = termoAritmetico();
        ret.adicionarFilho(termo);
        return ret;
    } else {
        erroSintatico("+", "-");
    }
    return null;
}

```

```

//termoAritmetico : termoAritmetico '*' fatorAritmetico | termoAritmetico
 '/' fatorAritmetico | fatorAritmetico;
// também precisa fatorar à esquerda e eliminar recursão à esquerda
// termoAritmetico : fatorAritmetico termoAritmetico2
// termoAritmetico2 : ('*' fatorAritmetico | '/' fatorAritmetico)
termoAritmetico2 | <<vazio>>
AsaNo termoAritmetico() {
    AsaNo fator = fatorAritmetico();
    AsaNo termo2 = termoAritmetico2(fator);
    if (termo2 != null) {
        return termo2;
    }
    return fator;
}

AsaNo termoAritmetico2(AsaNo fator1) {
    if (lookahead(1).nome == TipoToken.OpAritMult || lookahead(1).nome ==
TipoToken.OpAritDiv) {
        AsaNo termo1 = termoAritmetico2SubRegral();
        termo1.adicionarFilhoEsquerda(fator1);
        AsaNo termo2 = termoAritmetico2(termo1);
        if(termo2 != null) {
            return termo2;
        }
        return termo1;
    } else { // vazio
        return null;
    }
}

AsaNo termoAritmetico2SubRegral() {
    if (lookahead(1).nome == TipoToken.OpAritMult) {
        AsaNo ret = new AsaNo("*");
        ret.token = lookahead(1);
        match(TipoToken.OpAritMult);
        AsaNo fator = fatorAritmetico();
        ret.adicionarFilho(fator);
        return ret;
    } else if (lookahead(1).nome == TipoToken.OpAritDiv) {
        AsaNo ret = new AsaNo("/");
        ret.token = lookahead(1);
        match(TipoToken.OpAritDiv);
        AsaNo fator = fatorAritmetico();
        ret.adicionarFilho(fator);
        return ret;
    } else {
        erroSintatico("*", "/");
    }
    return null;
}

```

```

//fatorAritmetico : NUMINT | NUMREAL | VARIAVEL | '(' expressaoAritmetica
')'
AsaNo fatorAritmetico() {
    if (lookahead(1).nome == TipoToken.NumInt) {
        AsaNo ret = new AsaNo("NumInt");
        ret.token = lookahead(1);
        match(TipoToken.NumInt);
        return ret;
    } else if (lookahead(1).nome == TipoToken.NumReal) {
        AsaNo ret = new AsaNo("NumReal");
        ret.token = lookahead(1);
        match(TipoToken.NumReal);
        return ret;
    } else if (lookahead(1).nome == TipoToken.Var) {
        AsaNo ret = new AsaNo("Var");
        ret.token = lookahead(1);
        match(TipoToken.Var);
        return ret;
    } else if (lookahead(1).nome == TipoToken.AbrePar) {
        match(TipoToken.AbrePar);
        AsaNo ret = expressaoAritmetica();
        match(TipoToken.FechaPar);
        return ret;
    } else {

erroSintatico(TipoToken.NumInt.toString(), TipoToken.NumReal.toString(), TipoToken.Var.toString(), "(");
    }
    return null;
}

//expressaoRelacional : expressaoRelacional operadorBooleano
termoRelacional | termoRelacional;
// Precisa eliminar a recursão à esquerda
// expressaoRelacional : termoRelacional expressaoRelacional2;
// expressaoRelacional2 : operadorBooleano termoRelacional
expressaoRelacional2 | <<vazio>>
AsaNo expressaoRelacional() {
    AsaNo termoRel = termoRelacional();
    AsaNo exprRel2 = expressaoRelacional2(termoRel);
    if(exprRel2 != null) {
        return exprRel2;
    }
    return termoRel;
}

AsaNo expressaoRelacional2(AsaNo termoRel1) {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
(lookahead(1).lexema.equals("E") || lookahead(1).lexema.equals("OU"))) {
        AsaNo opBool = operadorBooleano();

```

```

        AsaNo termoRel = termoRelacional();
        opBool.adicionarFilho(termoRel);
        opBool.adicionarFilho(termoRel1);
        AsaNo exprRel2 = expressaoRelacional2(opBool);
        if(exprRel2 != null) {
            return exprRel2;
        }
        return opBool;
    } else { // vazio
        return null;
    }
}

//termoRelacional : expressaoAritmetica OP_REL expressaoAritmetica | '('
expressaoRelacional ')';
AsaNo termoRelacional() {
    if (lookahead(1).nome == TipoToken.NumInt
        || lookahead(1).nome == TipoToken.NumReal
        || lookahead(1).nome == TipoToken.Var
        || lookahead(1).nome == TipoToken.AbrePar) {
        // Há um não-determinismo aqui.
        // AbrePar pode ocorrer tanto em expressaoAritmetica como em
(expressaoRelacional)
        // Tem uma forma de resolver este problema, mas não usaremos aqui
        // Vamos modificar a linguagem, eliminando a possibilidade
        // de agrupar expressões relacionais com parêntesis
        AsaNo expr1 = expressaoAritmetica();
        AsaNo opRel = opRel();
        AsaNo expr2 = expressaoAritmetica();
        opRel.adicionarFilho(expr1);
        opRel.adicionarFilho(expr2);
        return opRel;
    } else {

erroSintatico(TipoToken.NumInt.toString(), TipoToken.NumReal.toString(), TipoTo
ken.Var.toString(), "(");
    }
    return null;
}

AsaNo opRel() {
    AsaNo opRel = new AsaNo("opRel");
    opRel.nome = lookahead(1).lexema;
    if (lookahead(1).nome == TipoToken.OpRelDif) {
        match(TipoToken.OpRelDif);
    } else if (lookahead(1).nome == TipoToken.OpRelIgual) {
        match(TipoToken.OpRelIgual);
    } else if (lookahead(1).nome == TipoToken.OpRelMaior) {
        match(TipoToken.OpRelMaior);
    } else if (lookahead(1).nome == TipoToken.OpRelMaiorIgual) {
        match(TipoToken.OpRelMaiorIgual);
    }
}

```



```

    } else if (lookahead(1).nome == TipoToken.OpRelMenor) {
        match(TipoToken.OpRelMenor);
    } else if (lookahead(1).nome == TipoToken.OpRelMenorIgual) {
        match(TipoToken.OpRelMenorIgual);
    } else {
        erroSintatico("<>", "=", ">", ">=", "<", "<=");
    }
    return opRel;
}

//operadorBooleano : 'E' | 'OU';
AsaNo operadorBooleano() {
    AsaNo opBool = new AsaNo("opBool");
    opBool.token = lookahead(1);
    opBool.nome = opBool.token.lexema;
    if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("E")) {
        match(TipoToken.PalavraChave, "E");
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("OU")) {
        match(TipoToken.PalavraChave, "OU");
    } else {
        erroSintatico("E", "OU");
    }
    return opBool;
}

//listaComandos : comando listaComandos | comando;
// vamos fatorar à esquerda
// listaComandos : comando (listaComandos | <<vazio>>)
void listaComandos(AsaNo algoritmo) {
    AsaNo comandoNo = comando();
    algoritmo.adicionarFilho(comandoNo);
    listaComandosSubRegral(algoritmo);
}

void listaComandosSubRegral(AsaNo algoritmo) {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
(lookahead(1).lexema.equals("ATRIBUIR")
    || lookahead(1).lexema.equals("LER")
    || lookahead(1).lexema.equals("IMPRIMIR")
    || lookahead(1).lexema.equals("SE")
    || lookahead(1).lexema.equals("ENQUANTO")
    || lookahead(1).lexema.equals("INICIO"))) {
        listaComandos(algoritmo);
    } else {
        // vazio
    }
}

//comando : comandoAtribuicao | comandoEntrada | comandoSaida |

```

```

comandoCondicao | comandoRepeticao | subAlgoritmo;
AsaNo comando() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("ATRIBUIR")) {
        return comandoAtribuicao();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("LER")) {
        return comandoEntrada();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("IMPRIMIR")) {
        return comandoSaida();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("SE")) {
        return comandoCondicao();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("ENQUANTO")) {
        return comandoRepeticao();
    } else if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("INICIO")) {
        return subAlgoritmo();
    } else {

erroSintatico("ATRIBUIR","LER","IMPRIMIR","SE","ENQUANTO","INICIO");
        return null;
    }
}

//comandoAtribuicao : 'ATRIBUIR' expressaoAritmetica 'A' VARIABEL;
AsaNo comandoAtribuicao() {
    AsaNo cmd = new AsaNo("ATRIBUIR");
    match(TipoToken.PalavraChave, "ATRIBUIR");
    AsaNo expr = expressaoAritmetica();
    match(TipoToken.PalavraChave, "A");
    AsaNo var = new AsaNo("Var");
    Token varToken = lookahead(1);
    var.token = varToken;
    match(TipoToken.Var);
    cmd.adicionarFilho(expr);
    cmd.adicionarFilho(var);
    return cmd;
}

//comandoEntrada : 'LER' VARIABEL;
AsaNo comandoEntrada() {
    AsaNo cmd = new AsaNo("LER");
    match(TipoToken.PalavraChave, "LER");
    Token varToken = lookahead(1);
    cmd.token = varToken;
    match(TipoToken.Var);
    return cmd;
}

```

```

//comandoSaida : 'IMPRIMIR' (VARIABLE | CADEIA);
AsaNo comandoSaida() {
    AsaNo cmd = new AsaNo("IMPRIMIR");
    match(TipoToken.PalavraChave, "IMPRIMIR");
    AsaNo cmd1 = new AsaNo("IMPRESSO");
    cmd1.token = lookahead(1);
    comandoSaidaSubRegral();
    cmd.adicionarFilho(cmd1);
    return cmd;
}

void comandoSaidaSubRegral() {
    if (lookahead(1).nome == TipoToken.Var) {
        match(TipoToken.Var);
    } else if (lookahead(1).nome == TipoToken.Cadeia) {
        match(TipoToken.Cadeia);
    } else {

erroSintatico(TipoToken.Var.toString(), TipoToken.Cadeia.toString());
    }
}

//comandoCondicao : 'SE' expressaoRelacional 'ENTAO' comando | 'SE'
expressaoRelacional 'ENTAO' comando 'SENAO' comando;
// fatorar à esquerda
// comandoCondicao : 'SE' expressaoRelacional 'ENTAO' comando ('SENAO'
comando | <<vazio>>)
AsaNo comandoCondicao() {
    AsaNo cmd = new AsaNo("SE");
    match(TipoToken.PalavraChave, "SE");
    AsaNo exprSe = expressaoRelacional();
    match(TipoToken.PalavraChave, "ENTAO");
    AsaNo cmdEntao = comando();
    AsaNo cmdSenao = comandoCondicaoSubRegral();
    cmd.adicionarFilho(exprSe);
    cmd.adicionarFilho(cmdEntao);
    if(cmdSenao != null)
        cmd.adicionarFilho(cmdSenao);
    return cmd;
}

AsaNo comandoCondicaoSubRegral() {
    if (lookahead(1).nome == TipoToken.PalavraChave &&
lookahead(1).lexema.equals("SENAO")) {
        match(TipoToken.PalavraChave, "SENAO");
        return comando();
    } else {
        // vazio
        return null;
    }
}

```

```

}

//comandoRepeticao : 'ENQUANTO' expressaoRelacional comando;
AsaNo comandoRepeticao() {
    AsaNo cmd = new AsaNo("ENQUANTO");
    match(TipoToken.PalavraChave, "ENQUANTO");
    AsaNo expr = expressaoRelacional();
    AsaNo cmdRepeticao = comando();
    cmd.adicionarFilho(expr);
    cmd.adicionarFilho(cmdRepeticao);
    return cmd;
}

//subAlgoritmo : 'INICIO' listaComandos 'FIM';
AsaNo subAlgoritmo() {
    AsaNo cmd = new AsaNo("SUBALGORITMO");
    match(TipoToken.PalavraChave, "INICIO");
    listaComandos(cmd);
    match(TipoToken.PalavraChave, "FIM");
    return cmd;
}

```

### 10.3. Modificar o void main para obter e imprimir a árvore

```

AsaNo asa = parser.programa();
System.out.println(asa);

```

## 11. Testar

## Demonstração 2 – Analisador sintático preditivo de descendência recursiva – ANTLR

---

1. Abrir o NetBeans
2. Criar um novo projeto Java, chamado “AlgumaParserAntlr”
3. Criar uma nova gramática (combined grammar), chamada “Alguma”

```
grammar Alguma;
```

```
TIPO_VAR
```

```
    :    'INTEIRO' | 'REAL';
```

```
NUMINT
```

```
    :    ('0'..'9')+  
    ;
```

```
NUMREAL
```

```
    :    ('0'..'9')+ ('.' ('0'..'9')+)?  
    ;
```

```
VARIAVEL
```

```
    :    ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9')*  
    ;
```

```
CADEIA
```

```
    :    '\\' ( ESC_SEQ | ~('\\'|'\\') )* '\\'  
    ;
```

```
OP_ARIT1
```

```
    :    '+' | '-'  
    ;
```

```
OP_ARIT2
```

```
    :    '*' | '/'  
    ;
```

```
OP_REL
```

```
    :    '>' | '>=' | '<' | '<=' | '<>' | '='  
    ;
```

```
OP_BOOL
```

```
    :    'E' | 'OU'  
    ;
```

```
fragment
```

```
ESC_SEQ
```

```
    :    '\\\\';
```

```
COMENTARIO
```

```
    :    '%' ~('\n'|\r)* '\r'? '\n' {skip();}  
    ;
```

```

WS      :      ( ' ' | '\t' | '\r' | '\n') {skip();}
      ;

programa
      :      ':' 'DECLARACOES' listaDeclaracoes ':' 'ALGORITMO' listaComandos
      ;

listaDeclaracoes
      :      declaracao listaDeclaracoes | declaracao
      ;

declaracao
      :      VARIAVEL ':' TIPO_VAR
      ;

expressaoAritmetica
      :      expressaoAritmetica OP_ARIT1 termoAritmetico
      |      termoAritmetico
      ;

termoAritmetico
      :      termoAritmetico OP_ARIT2 fatorAritmetico
      |      fatorAritmetico
      ;

fatorAritmetico
      :      NUMINT
      |      NUMREAL
      |      VARIAVEL
      |      '(' expressaoAritmetica ')'
      ;

expressaoRelacional
      :      expressaoRelacional OP_BOOL termoRelacional
      |      termoRelacional
      ;

termoRelacional
      :      expressaoAritmetica OP_REL expressaoAritmetica
      |      '(' expressaoRelacional ')'
      ;

listaComandos
      :      comando listaComandos
      |      comando
      ;

comando
      :      comandoAtribuicao
      |      comandoEntrada

```

```

    | comandoSaida
    | comandoCondicao
    | comandoRepeticao
    | subAlgoritmo
;

comandoAtribuicao
: 'ATRIBUIR' expressaoAritmetica 'A' VARIAVEL
;

comandoEntrada
: 'LER' VARIAVEL
;

comandoSaida
: 'IMPRIMIR' (VARIAVEL | CADEIA)
;

comandoCondicao
: 'SE' expressaoRelacional 'ENTAO' comando
| 'SE' expressaoRelacional 'ENTAO' comando 'SENAO' comando
;

comandoRepeticao
: 'ENQUANTO' expressaoRelacional comando
;

subAlgoritmo
: 'INICIO' listaComandos 'FIM'
;

```

**4. IMPORTANTE:** em versões anteriores do ANTLR recursão à esquerda não era permitido, mas agora funciona

**5. Gerar código (especificar pacote "algumaparserantlr")**

**6. Adicionar biblioteca do ANTLR**

**7. Criar o seguinte código no main()**

```

    ANTLRInputStream input = new ANTLRInputStream(new
FileInputStream(<Caminho do arquivo>));
    AlgumaLexer lexer = new AlgumaLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    AlgumaParser parser = new AlgumaParser(tokens);
    parser.programa();

```

**8. Executar (vai dar erro na expressão booleana)**

**9. Fazer o debug do léxico**

9.1. Mostrar que está interpretando "E" e "OU" como variáveis

**10. Mover a regra OP\_BOOL para antes da regra VARIAVEL**

**11. Refazer os testes**

**12. Inserir alguns erros e testar**

12.1. Mostrar que alguns erros entre comandos ele não detecta

12.2. Utilizar a regra (comando)+ para lista de comandos

12.3. Fazer o mesmo para lista de declarações

## 12.4. Testar novamente

### 13. Inserir algumas ações para imprimir o que está reconhecendo

```
programa
:      { System.out.println("Começou um programa"); }
      ':' 'DECLARACOES'
      { System.out.println("  Declarações"); }
      listaDeclaracoes ':' 'ALGORITMO'
      { System.out.println("  Algoritmo"); }
      listaComandos
;
declaracao
:      VARIAVEL ':' TIPO_VAR
      { System.out.println("  Declaração: Var="+$VARIAVEL.text+",
Tipo="+$TIPO_VAR.text); }
;
comandoAtribuicao
:      'ATRIBUIR' expressaoAritmetica 'A' VARIAVEL
      { System.out.println("      "+$VARIAVEL.text+" =
"+$expressaoAritmetica.text); }
;
comandoEntrada
:      'LER' VARIAVEL
      { System.out.println("      "+$VARIAVEL.text+" = ENTRADA"); }
;
comandoSaida
:      'IMPRIMIR' texto=(VARIAVEL| CADEIA)
      { System.out.println("      IMPRIMIR "+$texto.text); }
;
```

## 14. Testar

### 15. Mostrar o uso de retorno

```
listaComandos : cmd=comando { System.out.println("Apareceu um comando do tipo
"+$cmd.tipoComando); } listaComandos
|      cmd=comando { System.out.println("Apareceu um comando do tipo
"+$cmd.tipoComando); };
```

```
comando returns [ String tipoComando ]
:      comandoAtribuicao { $tipoComando = "Atribuicao"; }
|      comandoEntrada { $tipoComando = "Entrada"; }
|      comandoSaida { $tipoComando = "Saida"; }
|      comandoCondicao { $tipoComando = "Condicao"; }
|      comandoRepeticao { $tipoComando = "Repeticao"; }
|      subAlgoritmo { $tipoComando = "Subalgoritmo"; };
```

## 16. Testar

### 17. Mostrar outro tipo de retorno

```
programa : ':' 'DECLARACOES' listaDeclaracoes ':' 'ALGORITMO'
lc=listaComandos { System.out.println("Numero de comandos:
```



```
 "+$lc.numComandos); };
```

```
listaComandos returns [ int numComandos ] : cmd=comando {  
System.out.println("Apareceu um comando do tipo "+$cmd.tipoComando); }  
lc=listaComandos { $numComandos = $lc.numComandos + 1;}  
    |      cmd=comando { System.out.println("Apareceu um comando do tipo  
"+$cmd.tipoComando); $numComandos = 1;};
```

## 18. Testar