Construção de Compiladores 1 - 2018.1 - Profs. Mário César San Felice (e Helena Caseli, Murilo Naldi, Daniel Lucrédio)
Tópico 07 - Análise Semântica - Roteiro de Códigos

Demonstração 1 – Analisador semântico "na mão"

---

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

```
: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 ENTAO
   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
ENTAO
   INICIO
      ATRIBUIR (numero3) A aux
      ATRIBUIR numero1 A numero3
      ATRIBUIR aux A numero1
   FIM
SE numero2 > numero3 ENTAO
   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 "AlgumaParser"
3. Copiar para projeto "AlgumaParserComAnalisadorSemantico"

4. Criar enumeration TipoVariavel

```
public enum TipoVariavel {
    INTEIRO, REAL
}
```

5. Criar classe EntradaTabelaDeSimbolos

```
public class EntradaTabelaDeSimbolos {
    public String nome;
    public TipoVariavel tipo;
}
```

6. Criar classe TabelaDeSimbolos

```
public class TabelaDeSimbolos {
    private List<EntradaTabelaDeSimbolos> tabelaDeSimbolos;

    public TabelaDeSimbolos() {
        tabelaDeSimbolos = new ArrayList<EntradaTabelaDeSimbolos>();
    }

    public int instalarNome(String nome, TipoVariavel tipo) {
        if(jaFoiDeclarado(nome)) {
            throw new RuntimeException("Erro semântico: Variável "+nome+" foi
declarada duas vezes");
        }
        EntradaTabelaDeSimbolos etds = new EntradaTabelaDeSimbolos();
        etds.nome = nome;
        etds.tipo = tipo;
        tabelaDeSimbolos.add(etds);
        return tabelaDeSimbolos.size()-1;
    }

    public TipoVariavel determinaTipo(String nome) {
        for(EntradaTabelaDeSimbolos etds:tabelaDeSimbolos) {
            if(etds.nome.equals(nome))
                return etds.tipo;
        }
        return null;
    }

    public boolean jaFoiDeclarado(String nome) {
        for(EntradaTabelaDeSimbolos etds:tabelaDeSimbolos) {
            if(etds.nome.equals(nome))
                return true;
        }
        return false;
    }
}
```

7. Adicionar o objeto da tabela de símbolos no parser

```
    TabelaDeSimbolos ts;

    public AlgumaParser(AlgumaLexico lex) {
```

```
        ts = new TabelaDeSimbolos();
      ...
```

8. Modificar a regra tipoVar

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

9. Modificar a regra declaracao

```
    //declaracao : VARIAVEL ':' tipoVar;
    void declaracao() {
        String nomeVar = lookahead(1).lexema;
        match(TipoToken.Var);
        match(TipoToken.Delim);
        TipoVariavel tipoVar = tipoVar();
        ts.instalarNome(nomeVar, tipoVar);
    }
```

10. Modificar as regras para expressões aritméticas

```
    TipoVariavel expressaoAritmetica() {
        TipoVariavel tipoTermo = termoAritmetico();
        TipoVariavel tipoExp = expressaoAritmetica2();
        if (tipoTermo == TipoVariavel.REAL || tipoExp == TipoVariavel.REAL) {
            return TipoVariavel.REAL;
        } else {
            return TipoVariavel.INTEIRO;
        }
    }

    TipoVariavel expressaoAritmetica2() {
        if (lookahead(1).nome == TipoToken.OpAritSoma || lookahead(1).nome ==
TipoToken.OpAritSub) {
            TipoVariavel tipoExp1 = expressaoAritmetica2SubRegra1();
            TipoVariavel tipoExp2 = expressaoAritmetica2();
            if (tipoExp1 == TipoVariavel.REAL || tipoExp2 ==
TipoVariavel.REAL) {
```

```
                return TipoVariavel.REAL;
            } else {
                return TipoVariavel.INTEIRO;
            }
        } else { // vazio
            return null;
        }
    }

    TipoVariavel expressaoAritmetica2SubRegra1() {
        if (lookahead(1).nome == TipoToken.OpAritSoma) {
            match(TipoToken.OpAritSoma);
            return termoAritmetico();
        } else if (lookahead(1).nome == TipoToken.OpAritSub) {
            match(TipoToken.OpAritSub);
            return termoAritmetico();
        } 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>>
    TipoVariavel termoAritmetico() {
        TipoVariavel tipoTermo1 = fatorAritmetico();
        TipoVariavel tipoTermo2 = termoAritmetico2();
        if (tipoTermo1 == TipoVariavel.REAL || tipoTermo2 ==
TipoVariavel.REAL) {
            return TipoVariavel.REAL;
        } else {
            return TipoVariavel.INTEIRO;
        }
    }

    TipoVariavel termoAritmetico2() {
        if (lookahead(1).nome == TipoToken.OpAritMult || lookahead(1).nome ==
TipoToken.OpAritMult) {
            TipoVariavel tipoTermo1 = termoAritmetico2SubRegra1();
            TipoVariavel tipoTermo2 = termoAritmetico2();
            if (tipoTermo1 == TipoVariavel.REAL || tipoTermo2 ==
TipoVariavel.REAL) {
                return TipoVariavel.REAL;
            } else {
                return TipoVariavel.INTEIRO;
            }
        } else { // vazio
```

```
            return null;
        }
    }

    TipoVariavel termoAritmetico2SubRegra1() {
        if (lookahead(1).nome == TipoToken.OpAritMult) {
            match(TipoToken.OpAritMult);
            return fatorAritmetico();
        } else if (lookahead(1).nome == TipoToken.OpAritDiv) {
            match(TipoToken.OpAritDiv);
            return fatorAritmetico();
        } else {
            erroSintatico("*", "/");
            return null;
        }
    }

    //fatorAritmetico : NUMINT | NUMREAL | VARIAVEL | '(' expressaoAritmetica
')'
    TipoVariavel fatorAritmetico() {
        if (lookahead(1).nome == TipoToken.NumInt) {
            match(TipoToken.NumInt);
            return TipoVariavel.INTEIRO;
        } else if (lookahead(1).nome == TipoToken.NumReal) {
            match(TipoToken.NumReal);
            return TipoVariavel.REAL;
        } else if (lookahead(1).nome == TipoToken.Var) {
            String nomeVar = lookahead(1).lexema;
            if (!ts.jaFoiDeclarado(nomeVar)) {
                throw new RuntimeException("Erro semântico: Variável " +
nomeVar + " não foi declarada!");
            }
            TipoVariavel tipoVar = ts.determinaTipo(nomeVar);
            match(TipoToken.Var);
            return tipoVar;
        } else if (lookahead(1).nome == TipoToken.AbrePar) {
            match(TipoToken.AbrePar);
            TipoVariavel tipoExp = expressaoAritmetica();
            match(TipoToken.FechaPar);
            return tipoExp;
        } else {
            erroSintatico(TipoToken.NumInt.toString(),
TipoToken.NumReal.toString(), TipoToken.Var.toString(), "(");
            return null;
        }
    }
```

11. Modificar a regra de atribuição

```
    void comandoAtribuicao() {
        match(TipoToken.PalavraChave, "ATRIBUIR");
```

```
        TipoVariavel tipoExp = expressaoAritmetica();
        match(TipoToken.PalavraChave, "A");
        String nomeVar = lookahead(1).lexema;
        if (!ts.jaFoiDeclarado(nomeVar)) {
            throw new RuntimeException("Erro semântico: Variável " + nomeVar
+ " não foi declarada!");
        }
        TipoVariavel tipoVar = ts.determinaTipo(nomeVar);
        match(TipoToken.Var);
        if(tipoVar == TipoVariavel.INTEIRO && tipoExp == TipoVariavel.REAL)
            throw new RuntimeException("Erro semântico: Variável " + nomeVar
+ " deve ser do tipo REAL!");
    }
```

12. Comentar as linhas com System.out (Dentro dos métodos lerToken e match 2x)
13. Testar
    13.1. No programa de exemplo, vai dar erro que a variável numero4 não foi declarada
14. Modificar o programa para incluir um erro de atribuição

Ex: `ATRIBUIR numero4 A numero1`

Demonstração 2 – Analisador semântico no ANTLR

---

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

```
(34 - 3 + 2) * (41 + 3)
```

2. Criar um novo projeto NetBeans, chamado ExprParser
3. Criar um arquivo, dentro do projeto (pasta src, pacote exprparser) chamado Expressoes.g4

```
grammar Expressoes;

programa returns [ int val ]
      :     expressao EOF { $val = $expressao.val; }
      ;

expressao returns  [ int val ]
      :     termo expressao2[$termo.val] { $val = $expressao2.sint; }
      ;

expressao2 [ int her ] returns [ int sint ]
      :     '+' termo exp=expressao2[$termo.val+$her] { $sint = $exp.sint; }
      |     '-' termo exp=expressao2[$her-$termo.val] { $sint = $exp.sint; }
      |     { $sint = $her; }
      ;

termo returns [ int val ]
      :     fator termo2[$fator.val] { $val = $termo2.sint; }
      ;

termo2 [ int her ] returns [ int sint ]
      :     '*' fator term=termo2[$fator.val*$her] { $sint = $term.sint; }
      |     { $sint = $her; }
      ;

fator returns [ int val ]
      :     '(' expressao ')' { $val = $expressao.val; }
      |     NUM { $val = Integer.parseInt($NUM.getText()); }
      ;

NUM   :     '0'..'9'+
      ;

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

4. Compilar a gramática (não esquecer da opção -package exprparser)) e rodar com o seguinte método main

```
            ANTLRInputStream input = new ANTLRInputStream(new

FileInputStream("C:\\Users\\Daniel\\Desktop\\teste.txt"));
```

```
            ExpressoesLexer lexer = new ExpressoesLexer(input);
            CommonTokenStream tokens = new CommonTokenStream(lexer);
            ExpressoesParser parser = new ExpressoesParser(tokens);
            int val = parser.programa().val;
            System.out.println("Valor = "+val);
```

5. Agora vamos fazer de um jeito mais simples
6. Criar um novo projeto no NetBeans, chamado ExprParser2
7. Criar um arquivo, dentro do projeto (pasta src, pacote exprparser2) chamado Expressoes.g4

```
grammar Expressoes;
programa:
    expressao EOF
;
expressao:
    termo1=termo (op1 outrosTermos+=termo)*
;
termo:
    fator1=fator (op2 outrosFatores+=fator)*
;
fator:
    '(' expressao ')' |
    NUM
;
op1:
    '+' | '-'
;
op2:
    '*' | '/'
;
NUM:
    '0'..'9'+
;
WS:
    ( ' ' | '\n' | '\r' | '\t' ) -> skip
;
```

8. Compilar com a opção para gerar o visitor (-visitor) e não esquecer do pacote (-package exprparser2)
9. Criar uma classe Calculador

```
public class Calculador extends ExpressoesBaseVisitor<Double> {

    @Override
    public Double visitPrograma(ExpressoesParser.ProgramaContext ctx) {
        return visitExpressao(ctx.expressao());
    }

    @Override
    public Double visitExpressao(ExpressoesParser.ExpressaoContext ctx) {
        double valor = visitTermo(ctx.termo1);
```

```
        for(int i=0;i<ctx.outrosTermos.size();i++) {
            ExpressoesParser.Op1Context op1 = ctx.op1(i);
            TermoContext ot = ctx.outrosTermos.get(i);
            String strOp1 = op1.getText();
            if(strOp1.equals("+")) {
                valor += visitTermo(ot);
            } else {
                valor -= visitTermo(ot);
            }
        }
        return valor;
    }

    @Override
    public Double visitTermo(ExpressoesParser.TermoContext ctx) {
        double valor = visitFator(ctx.fator1);
        for(int i=0;i<ctx.outrosFatores.size();i++) {
            ExpressoesParser.Op2Context op2 = ctx.op2(i);
            FatorContext of = ctx.outrosFatores.get(i);
            String strOp2 = op2.getText();
            if(strOp2.equals("*")) {
                valor *= visitFator(of);
            } else {
                valor /= visitFator(of);
            }
        }
        return valor;
    }

    @Override
    public Double visitFator(ExpressoesParser.FatorContext ctx) {
        if(ctx.NUM() != null) {
            return Double.parseDouble(ctx.NUM().getText());
        } else {
            return visitExpressao(ctx.expressao());
        }
    }
}
```

10. Executar com o seguinte código main

```
        ANTLRInputStream input = new ANTLRInputStream(new
FileInputStream("/Users/daniellucredio/Desktop/exemplo.txt"));
        ExpressoesLexer lexer = new ExpressoesLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        ExpressoesParser parser = new ExpressoesParser(tokens);
        ExpressoesParser.ProgramaContext arvore = parser.programa();
        Calculador c = new Calculador();
        double val = c.visitPrograma(arvore);
        System.out.println("Valor2 = " + val);
```

Demonstração 3 – Uso da tabela de símbolos

---

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

```
let x=2+1, y=3+4 in x+y
```

2. Abrir o NetBeans, e criar um novo projeto Java “AnalisadorExpressoesAritmeticas”
3. Criar classe analisadorexpressoesaritmeticas.EntradaTabelaDeSimbolos

```
public class EntradaTabelaDeSimbolos {
    public String nome;
    public double valor;
}
```

4. Criar classe analisadorexpressoesaritmeticas.TabelaDeSimbolos

```
public class TabelaDeSimbolos {

    private HashMap<String, EntradaTabelaDeSimbolos> tabelaDeSimbolos;

    public TabelaDeSimbolos() {
        tabelaDeSimbolos = new HashMap<>();
    }

    public void inserir(String nome, double valor) {
        EntradaTabelaDeSimbolos etds = new EntradaTabelaDeSimbolos();
        etds.nome = nome;
        etds.valor = valor;
        tabelaDeSimbolos.put(nome, etds);
    }

    public EntradaTabelaDeSimbolos verificar(String nome) {
        if(!tabelaDeSimbolos.containsKey(nome))
            return null;
        else return tabelaDeSimbolos.get(nome);
    }
}
```

5. Criar classe analisadorexpressoesaritmeticas.Escopos

```
public final class Escopos {
    private LinkedList<TabelaDeSimbolos> pilhaDeTabelas;

    public Escopos() {
        pilhaDeTabelas  = new LinkedList<>();
        criarNovoEscopo();
    }

    public void criarNovoEscopo() {
        pilhaDeTabelas.push(new TabelaDeSimbolos());
    }
```

```
    public TabelaDeSimbolos pegarEscopoAtual() {
        return pilhaDeTabelas.peek();
    }

    public List<TabelaDeSimbolos> percorrerEscoposAninhados() {
        return pilhaDeTabelas;
    }

    public void abandonarEscopo() {
        pilhaDeTabelas.pop();
    }
}
```

6. Criar o seguinte código na classe principal

```
        ANTLRInputStream input = new ANTLRInputStream(new
FileInputStream("/Users/daniellucredio/Desktop/exemplo.txt"));
        ExpressoesAritmeticasLexer lexer = new
ExpressoesAritmeticasLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        ExpressoesAritmeticasParser parser = new
ExpressoesAritmeticasParser(tokens);
        ProgramaContext arvore = parser.programa();
        Calculador c = new Calculador();
        double val = c.visitPrograma(arvore);
        System.out.println("Valor calculado: "+val);
```

7. Abrir o ANTLRWorks e criar a seguinte gramática

```
grammar ExpressoesAritmeticas;

programa:
    exp
;

exp:
    termo1=termo ('+' outrosTermos+=termo)*
;

termo:
    '(' expParentesis=exp ')' |
    variavel=ID |
    constante=INT |
    'let' listaDecl 'in' subexp=exp
;

listaDecl:
    decl (',' decl)*
;
```

```
decl:
    nome=ID '=' exp
;

ID  : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*;
INT : '0'..'9'+;
WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        ) -> skip
;
```

8. Compilar com a opção de gerar o visitor, e criar a seguinte classe

```
public class Calculador extends ExpressoesAritmeticasBaseVisitor<Double> {

    Escopos escoposAninhados = new Escopos();

    @Override
    public Double visitPrograma(ExpressoesAritmeticasParser.ProgramaContext
ctx) {
        return visitExp(ctx.exp());
    }

    @Override
    public Double visitExp(ExpressoesAritmeticasParser.ExpContext ctx) {
        double valor = visitTermo(ctx.termo1);
        for (TermoContext ot : ctx.outrosTermos) {
            valor += visitTermo(ot);
        }
        return valor;
    }

    @Override
    public Double visitTermo(ExpressoesAritmeticasParser.TermoContext ctx) {
        if (ctx.expParentesis != null) {
            return visitExp(ctx.expParentesis);
        } else if (ctx.constante != null) {
            return Double.parseDouble(ctx.constante.getText());
        } else if (ctx.variavel != null) {
            List<TabelaDeSimbolos> escopos =
escoposAninhados.percorrerEscoposAninhados();
            for (TabelaDeSimbolos ts : escopos) {
                EntradaTabelaDeSimbolos etds =
ts.verificar(ctx.variavel.getText());
                if (etds != null) {
                    return etds.valor;
                }
            }
            throw new RuntimeException("Erro semântico: "
```

```
                    + ctx.variavel.getText() + " não foi declarada antes do
uso");
        } else {
            escoposAninhados.criarNovoEscopo();
            visitListaDecl(ctx.listaDecl());
            double retorno = visitExp(ctx.subexp);
            escoposAninhados.abandonarEscopo();
            return retorno;
        }
    }

    @Override
    public Double visitDecl(ExpressoesAritmeticasParser.DeclContext ctx) {
        TabelaDeSimbolos escopoAtual = escoposAninhados.pegarEscopoAtual();
        if (escopoAtual.verificar(ctx.nome.getText()) != null) {
            throw new RuntimeException("Erro semântico: " +
ctx.nome.getText()
                    + " declarada duas vezes num mesmo escopo");
        } else {
            escopoAtual.inserir(ctx.nome.getText(), visitExp(ctx.exp()));
        }

        return null; // declaração não tem valor
    }

    @Override
    public Double visitListaDecl(ExpressoesAritmeticasParser.ListaDeclContext
ctx) {
        for (DeclContext d : ctx.decl()) {
            visitDecl(d);
        }
        return null; // declaração não tem valor
    }

}
```

9. Testar com vários programas

```
let x=2+1, y=3+4 in x+y // valor=10

let x=2, y=3 in
    (let x=x+1, y=(let z=3, x=4 in x+y+z)
     in (x+y)
    ) // valor=13

let x=2,y=x+1 in (let x=x+y,y=x+y in y) // valor=8

let x=2,x=3 in x+1 // x declarada duas vezes

let x=2 in x+y // y não declarada
```

```
let x=2 in (let x=3 in x) //valor=3

let x=2+1, y=(let z=3 in z+x) in let z=4 in x+y+z //valor=13
```