# Adaptado de Construção de Compiladores 1 - 2013.1 Prof. Daniel Lucrédio
# Aula 04 - Análise Sintática Descendente - roteiro

## Instalando NetBeans e ANTLR

1. Instalar o NetBeans 7.3 seguindo as instruções em https://netbeans.org/downloads/ (Eu instalei a versão "Tudo" personalizando para não instalar suporte para Groovy nem PHP)

2. Adicionar o ANTLRWorks no NetBeans
    - Abrir o NetBeans 7.3
    - Ir no Menu "Ferramentas → Plug-ins", Aba "Definições", Botão "Adicionar"
        - Em Nome inserir ANTLRWorks e em URL adicionar http://tunnelvisionlabs.com/downloads/nbupdates/nb73/aw2/updates.xml (segundo instruções disponíveis em http://tunnelvisionlabs.com/products/demo/antlrworks)
    - Na Aba "Plug-ins Disponíveis" selecionar todos da categoria "Tunnel Vision Labs" e clicar no botão "Instalar"

3. Baixar aquivo com a biblioteca completa do ANTLR (versão "standalone") e salvar no seu computador
    - Acessar http://www.antlr.org/download.html
    - Clicar em Complete ANTLR 4.0 Java binaries jar

## Criando um Analisador sintático preditivo de descendência recursiva usando o ANTLR

1. Criar um novo arquivo texto .txt, no Desktop, com um programa de exemplo da linguagem Alguma (copiar-colar linhas abaixo):

```
: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 7.3

3. Criar um novo projeto Java (Aplicação Java), chamado “AlgumaParserAntlr”

4. Vincular a versão “standalone” do ANTLR ao projeto
- Navegar no projeto “AlgumaParserAntlr” até “Bibliotecas”
- Clicar com o botão direito do mouse em “Bibliotecas” e selecionar “Adicionar Biblioteca”
- Clicar no botão “Criar”
- Em Nome da Biblioteca inserir “ANTLR” e clicar em “OK”
- Clicar no botão “Adicionar JAR/Pasta”, navegar até o local onde você salvou a versão “standalone” do ANTLR e selecionar o arquivo antlr-4.0-complete.jar
- Clicar no botão “Adicionar JAR/Pasta”
- Clicar no botão “OK”
- Clicar no botão “Adicionar Biblioteca”

5. Criar uma nova gramática (combined grammar), chamada “Alguma”
- Clicar com o botão direito do mouse sobre o pacote “algumaparserantlr”, selecionar a opção “Novo → Outros”, selecionar “ANTLR → ANTLR 4 Combined Grammar”
- Em Nome do Arquivo inserir “Alguma”
- Clicar em “Finalizar”
- Copiar-colar a gramática abaixo no arquivo gerado Alguma.g4

```
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'
      ;
```

Vejam que na versão atual do ANTLR ele é capaz de lidar com gramáticas com recursão à esquerda e não fatoradas, ou seja, que não são LL(1). Isso porque a versão atual adota uma estratégia de verificar vários tokens à frente.

6. Gerar código (arquivos .tokens, Lexer e Parser)
- Com Alguma.g4 selecionado e aberto, ir no Menu “Executar → Generate Recognizer”
- Clicar em “Próximo”
- Desmarcar “Generate listener”
- Marcar “Package” e digitar o nome do pacote "algumaparserantlr"
- Clicar em “Próximo”
- Clicar em “Finalizar”
- Verificar grafos sintáticos no menu “Janela → Syntax Diagram” navegando pelas regras

7. Criar o seguinte código no main() com throws Exception

```
        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. Resolver pendência selecionando a opção “Adicionar importação para org.antlr.v4.runtime.ANTLRInputStream”

9. Alterar o <Caminho do arquivo> para o local onde o arquivo .txt criado no passo 1 foi salvo

10. Compilar e Executar (vai dar erro na expressão booleana porque está interpretando "E" e "OU" como variáveis)

11. Fazer o debug do léxico
- Com Alguma.g4 selecionado e aberto, ir no Menu “Executar → Interpret Lexer”
- Selecionar o arquivo .txt criado no passo 1
- Ir no Menu “Janela → Lexer Debugger Controller”
- Veja em @128 que o 'E' foi identificado como VARIAVEL devido ao erro corrigido no passo a seguir

12. Mover a regra OP_BOOL para antes da regra VARIAVEL

13. Gerar código (passo 6 necessário toda vez que Alguma.g4 for alterado), Compilar e Executar novamente (agora não vai dar erro)

14. 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); }
      ;
```

15. Gerar código (passo 6), Compilar e Executar verificando as mensagens impressas na saída

16. Inserir algumas ações semânticas com valor 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"; };
```

17. Gerar código (passo 6), Compilar e Executar verificando novas mensagens impressas na saída

18. Inserir outros tipos 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;};
```

19. Gerar código (passo 6), Compilar e Executar verificando novas mensagens impressas na saída