INF441 - Roteiro para Aula Prática
Analisador Sintático usando ANTLR

Faça download do arquivo usando o link a seguir, que contém um projeto Eclipse compactado:

https://drive.google.com/file/d/1qXBDYosH84Gk2QNRz1ikgy3Ht0o-janM/view?usp=sharing

Em seguida, descompacte o conteúdo em uma pasta qualquer, em um computador que tenha Eclipse + ANTLR4 instalado.

Abra o IDE Eclipse e importe o projeto descompactado. Antes de iniciar este roteiro, verifique se o IDE Eclipse está devidamente configurado, seguindo as instruções em:

https://drive.google.com/file/d/1TWBsuBXlzZmSvHniZQ80POzoCVSaPbhP/view?usp=sharing

Em seguida, siga as instruções do roteiro para atividades relacionadas à construção de um analisador sintático usando ANTLR.

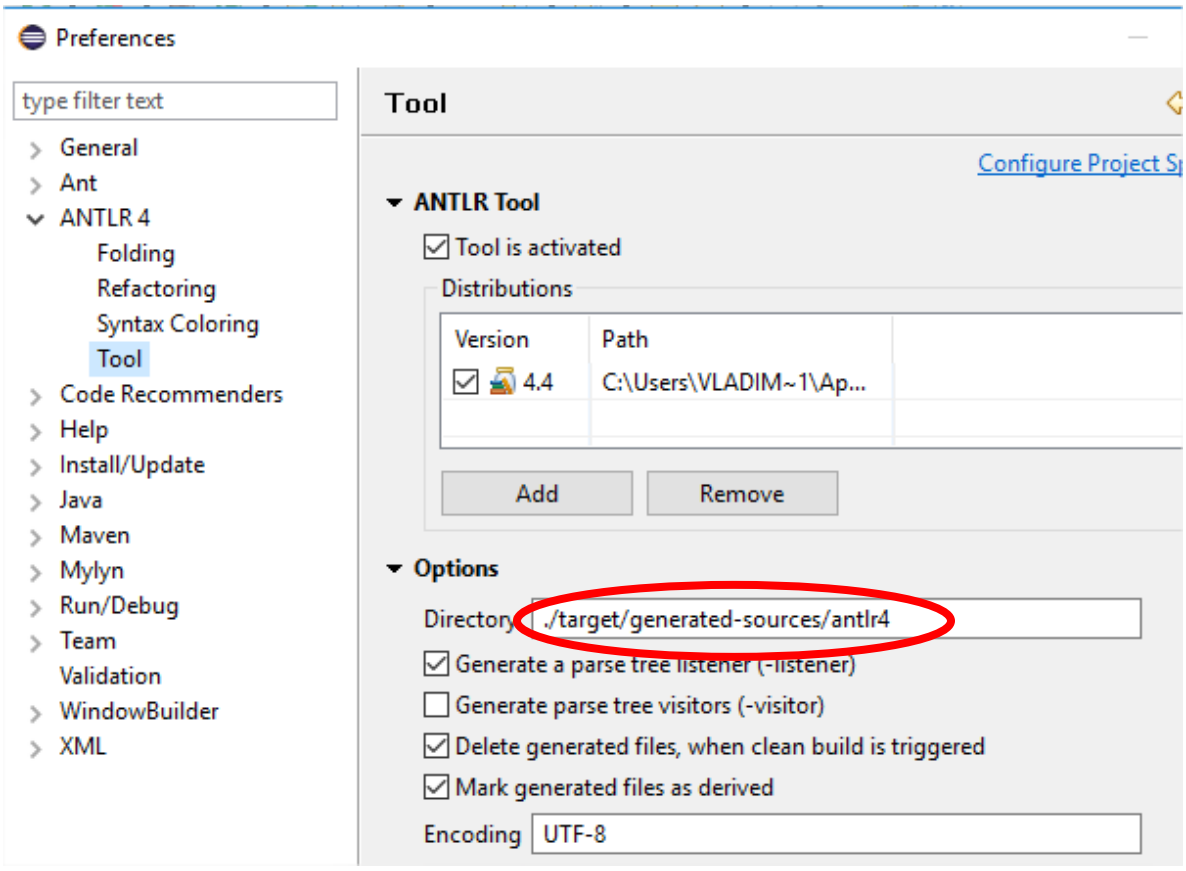
Diretivas Iniciais

Observe a especificação de uma gramática apresentada em *grammars/MicroJava.g4*, escrita em formato ANTLR. A especificação inicia com

```
grammar MicroJava;
```

Essa declaração indica que se trata de uma especificação que pode combinar analisador léxico e sintático, cujo nome é MicroJava. A partir dessa especificação, o ANTLR gera um analisador léxico e um analisador sintático. Observe esses analisadores como classes Java, na pasta *target/generated-sources/antlr4/srcparser*.

A pasta *target/generated-sources/antlr4* é definida como a pasta onde os analisadores serão gerados, para todos os projetos de um determinado workspace. Essa informação pode ser observada e alterada no menu Window-Preferences.



Dentro da pasta target/generated-sources/antlr4, a pasta *srcparser* é um package Java, definido pela diretiva

```
@header {
package srcparser;
}
```

que está especificada no arquivo *MicroJava.g4*. A seção *@header* indica o que pode aparecer no cabeçalho dos analisadores gerados. Caso um dos comandos seja *package*, como no caso acima, a pasta do pacote é criada e os analisadores são gerados nessa pasta.

Produções da Gramática

As produções de uma gramática ANTLR podem ser regras para o analisador léxico ou para o analisador sintático. As regras têm o formato geral

```
identificador : definição_da_regra ;
```

Se o identificador iniciar com letra maiúscula, é uma regra para o analisador léxico; se iniciar com letra minúscula, é uma regra para o analisador sintático.

O arquivo *grammars/MicroJava.g4* apresenta uma série de regras para o analisador léxico, listadas logo após o comentário "Lexer Definitions". As regras para o analisador sintático iniciam após o comentário "Parser Definitions".

A linguagem especificada pela gramática é descrita abaixo, supondo `program` como símbolo inicial. A notação usada é EBNF, onde o meta-símbolo `|` significa escolha entre alternativas, **`{...}`** significa 0 ou mais repetições, e **`[...]`** significa opcional (0 ou 1 ocorrência). Essa notação é a mesma usada pelo documento que descreve a sintaxe da linguagem MicroJava, disponibilizado no sítio da disciplina no PVANET.

```
program -> "program" IDENT {methodDecl}
typeOrVoid -> IDENT | "void"
methodDecl -> typeOrVoid IDENT "(" ")" block
block -> "{" {statement} "}"
statement -> IDENT "=" expr ";" | "return" [expr] ";" | block
expr -> ["-"] term {addOp term}
term -> factor {mulOp factor}
factor -> NUMBER | IDENT | "(" expr ")"
addOp -> "+" | "-"
mulOp -> "*" | "/" | "%"
```

Na gramática descrita, `IDENT` e `NUMBER` são terminais que representam, respectivamente, identificadores e constantes inteiras. A gramática define programas simples formados por uma sequência de funções sem parâmetros. Dentro de cada função, são permitidos apenas comandos de atribuição e comando *return*, com valor de retorno opcional. Não há declaração de variáveis.

Uma forma sentencial válida pode ser:

```
program P

int f() {
  a = -b + c * d;
  return a;
}
```

Traduzir uma gramática em formato EBNF para o formato ANTLR é simples. Em ANTLR, a escolha entre alternativas é representada pelo mesmo meta-símbolo `|`, repetições são representadas por `*` e opcionais são representados por `?`. Observe as regras apresentadas no arquivo e compare-as com as da gramática acima.

Teste do Analisador

O analisador sintático gerado usa a técnica de análise sintática recursiva descendente. Cada não terminal dá origem a uma função, que executa chamadas às funções associadas aos símbolos de suas produções. Para executar o analisador a partir do símbolo inicial da gramática, deve ser chamado o método associado a esse símbolo.

Observe o código de um programa piloto, em *Piloto3.java*:

```
public class Piloto3 {

	public static void testeParser(String fileName) throws Exception {
		ANTLRInputStream input = new ANTLRFileStream(fileName);
		MicroJavaLexer lexer = new MicroJavaLexer(input);
		CommonTokenStream tokens = new CommonTokenStream(lexer);
		MicroJavaParser parser = new MicroJavaParser(tokens);
		ParseTree tree = parser.program();
		System.out.println(tree.toStringTree(parser));
	}

	public static void main(String args[]) throws Exception {
		testeParser("input/teste01.txt");
	}
}
```

Inicialmente, é aberto um arquivo cujo nome é fornecido, e que irá obter sequência de caracteres da entrada. Esse arquivo é usado pelo analisador léxico *MicroJavaLexer*. Um gerenciador de sequência de tokens usa os resultados produzidos pelo analisador léxico, e o analisador sintático *MicroJavaParser* usa essa sequência de tokens. Observe que o método *program* é chamado, indicando que a função associada ao símbolo inicial da gramática (*program*) será executada. Uma árvore associada ao resultado é obtida e impressa.

Teste 1

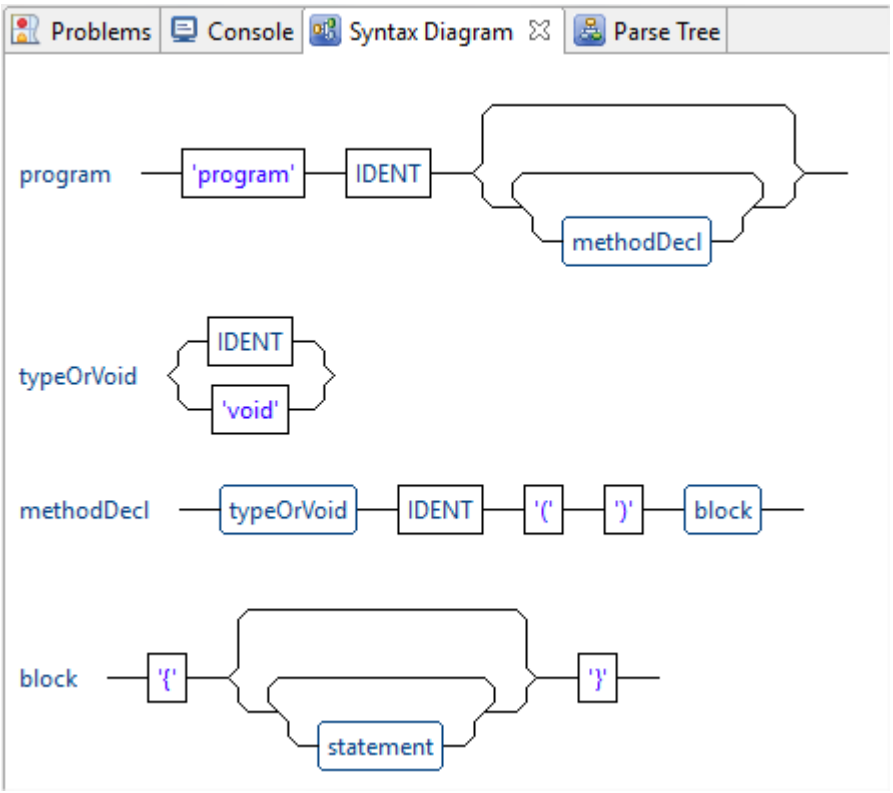
Execute a classe *Piloto3* e observe o resultado. O arquivo de entrada é lido e processado pelo analisador sintático, gerando uma árvore de sintaxe. Essa árvore é impressa em um formato textual, onde os nós filhos são representados por elementos dentro de parêntesis.

Faça alterações no arquivo de entrada *input/teste01.txt* e rode novamente a classe Piloto3. Experimente introduzir erros de sintaxe no arquivo e observe as mensagens de erro.

Teste 2

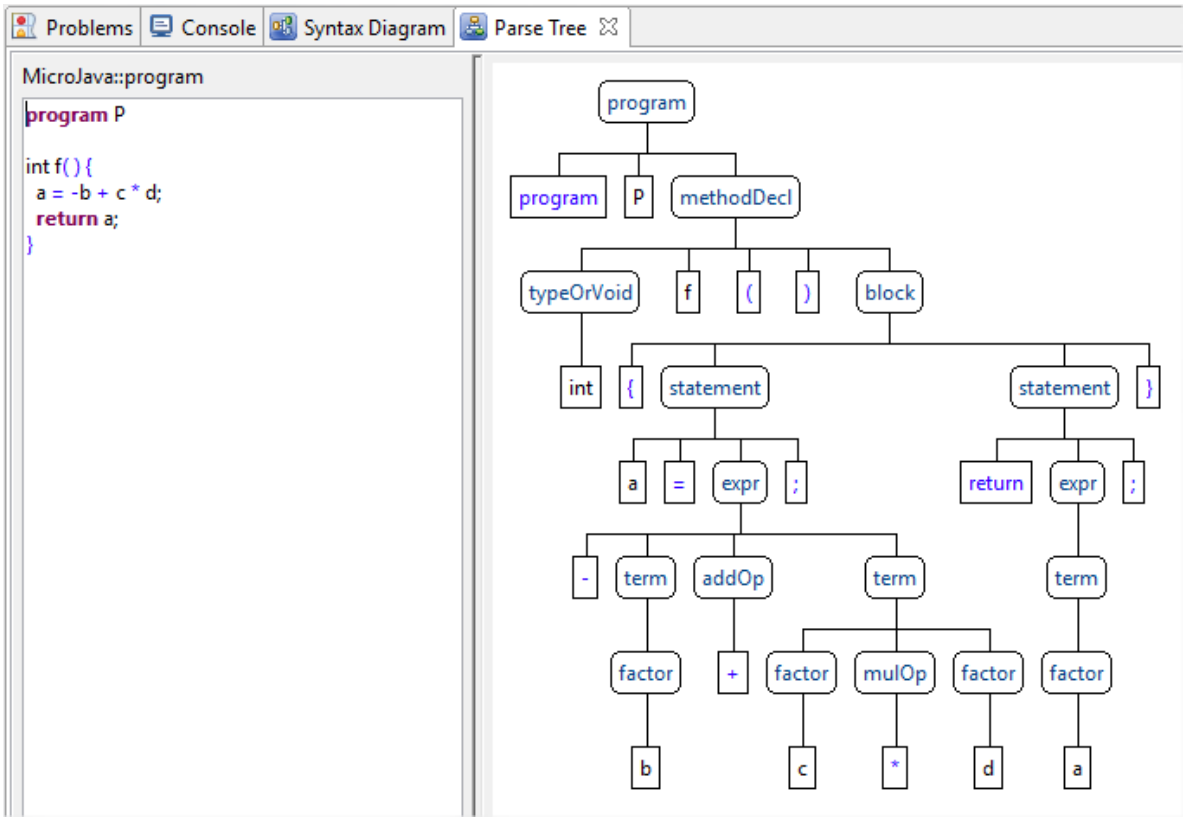
Abra as *views* específicas do plugin ANTRL4: “Syntax Diagram” e “Parse Tree”. Para isso, use as opções: Window - Show View - Other - ANTLR4

Observe a representação da gramática na *view* Syntax Diagram. Para isso, pode ser necessário abrir a *view* e depois modificar e gravar novamente a gramática MicroJava. Compare a representação gráfica com a definição original da gramática e procure entender a semântica da representação. Ela poderá ser útil quando você for estender a gramática.



Abra simultaneamente o arquivo *MicroJava.g4* e a *view* Parse Tree. No arquivo *MicroJava.g4*, clique na regra que define o não terminal *program*. No cabeçalho da janela da esquerda da *view* Parse Tree, deverá aparecer uma mensagem indicando que a regra

program foi selecionada. Em seguida, copie o conteúdo do arquivo input/teste01.txt para essa janela e observe a árvore de sintaxe associada sendo exibida na janela da direita.



Essa ferramenta online é muito útil para verificar se as árvores estão sendo construídas corretamente, seguindo as regras de precedência e associatividade desejadas. Veja, por exemplo, que na expressão `-b+c*d`, os elementos envolvidos na multiplicação estão reunidos em uma mesma sub árvore, separada da adição. Essa disposição da árvore pode facilitar o processamento do programa em etapas posteriores.

Atividade 1

Defina um comando condicional *if* para a linguagem, inicialmente sem cláusula *else*:

```
statement :
  ...
  | "if" "(" condition ")" statement
  ;

condition: expr relOp expr ;

relOp : ... ;
```

Na regra *relOp*, defina todos os operadores geralmente usados em comparação de valores. Para gerar um novo analisador, basta gravar a especificação na IDE Eclipse. Execute testes para verificar se o analisador reconhece o novo comando. Faça testes com comandos condicionais aninhados (condicionais dentro de outros condicionais). Observe que a recursividade nas definições permite esse aninhamento. Verifique se as árvores sintáticas construídas têm formato esperado.

Atividade 2

Acrescente uma cláusula *else* (opcional) na definição. Depois grave o arquivo para gerar um novo analisador.

```
statement :
  ...
  | "if" "(" condition ")" statement ("else" statement)?
  ;
```

A nova gramática definida é ambígua! Para comprovar isso, tente encontrar 2 árvores de sintaxe diferentes que podem ser geradas para um mesmo comando como:

```
  if (a < d) if (d > 0) d = 0; else a = 0;
```

A dúvida gerada consiste em definir a qual comando *if* a cláusula *else* está associada. Esse é um problema muito comum na definição de sintaxe de linguagens de programação. A ferramenta ANTLR4 gera um analisador sintático que realiza opção por uma das árvores de sintaxe possíveis, sem emitir nenhum aviso de ambiguidade. A árvore de sintaxe escolhida pelo ANTLR é aquela em que cláusulas *else* são reunidas em uma mesma sub árvore que o último comando *if* utilizado. Essa opção facilita processar o comando de uma maneira que representa mais diretamente a semântica de linguagens como C, C++ e Java.

Teste comandos como o condicional apresentado acima, digitando-os na *view* Parse Tree. Verifique se as árvores de sintaxe têm o formato esperado.

Atividade 3

Modifique a gramática de modo a aceitar declarações de variáveis locais nas funções, de maneira similar à linguagem MicroJava. Devem ser permitidos programas como:

```
program P

int f()
  int a, b;
  float c, d, e;
{
  a = b * (c - 1);
  d = -a + e / 2;
  if (a < d) if (d > 0) d = 0; else a = 0;
  return d;
}
```