**Universidade Anhembi Morumbi**
**Teoria da computação e compiladores**

# LUMIN

Iago Pisa Bandeira - 12523179797

Paola Sthephany Ferreira Silva - 12523173830

## Gramática – Regras e Tokens Principais da Linguagem Lumin

A linguagem **Lumin** foi desenvolvida utilizando a ferramenta ANTLR 4. Sua gramática define uma estrutura simples para programas que envolvem variáveis, controle de fluxo, expressões aritméticas e booleanas, entrada e saída de dados, respeitando a precedência correta entre operadores.

## Regras Principais (Parser):

- **programa**: representa o início do programa, composto por uma sequência de comandos.
- **comando**: define os tipos possíveis de instruções — declaração, atribuição, condicional, repetição, leitura ou escrita.
- **declaracao**: define uma variável com um tipo e nome. Exemplo: ***clarus x;***
- **tipo**: representa os três tipos disponíveis: ***clarus*** (inteiro), ***flux*** (real) e ***veritas*** (booleano).
- **atribuicao**: atribui valores ou expressões a uma variável existente. Ex: ***x = 10 + 5;***
- **condicional**: estrutura ***se (...) { ... } senao { ... }***, para tomada de decisão.
- **repeticao**: estrutura de repetição ***enquanto (...) { ... }***.
- **leitura**: comando ***captura(...)*** para entrada de dados do usuário.
- **escrita**: comando ***eco(...)*** para exibir dados na tela.
- **bloco**: define um bloco de comandos entre ***{ ... }.***

### Expressão

Regras que suportam expressões com operadores de:

- multiplicação e divisão (*, /).
- adição e subtração (+, -).
- operadores relacionais (>, <, ==, !=).

- parênteses para agrupamento.
- constantes numéricas, booleanas e identificadores.

---

## Tokens Principais (Lexer):

### Tipos e Valores Lógicos

- **clarus:** inteiro
- **flux:** real
- **veritas:** booleano
- **verum:** verdadeiro
- **falsum**: falso

### Controle de Fluxo

- **se, senao:** estrutura condicional
- **enquanto:** laço de repetição

### Entrada e Saída

- **eco:** imprime valor na tela
- **captura:** lê valor do teclado

### Símbolos e Operadores

- Aritméticos: +, -, *, /
- Relacionais: >, <, ==, !=
- Delimitadores: (, ), {, }, ;, =

**Literais e Identificadores**

- **NUM_INT**: números inteiros ([0-9]+)
- **NUM_REAL:** números reais ([0-9]+.[0-9]+)
- **ID:** identificadores válidos ([a-zA-Z_][a-zA-Z0-9_]*)

**Espaços em Branco**

- **WS**: ignora espaços, tabs e quebras de linha ([ \t\r\n]+ -> skip;)

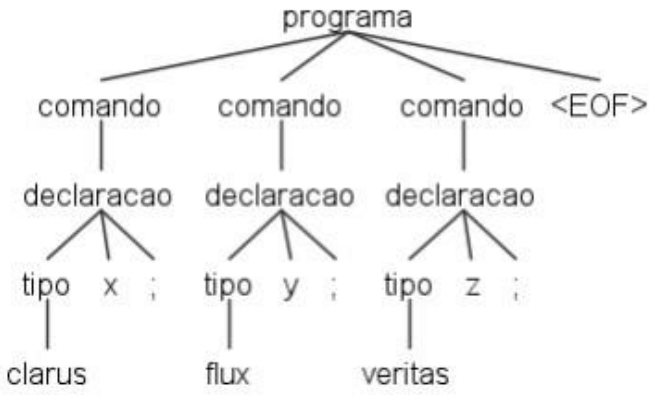
---

# Exemplos de Código Válido e Árvores de Análise Sintática

## Declaração de variáveis

clarus x;  flux

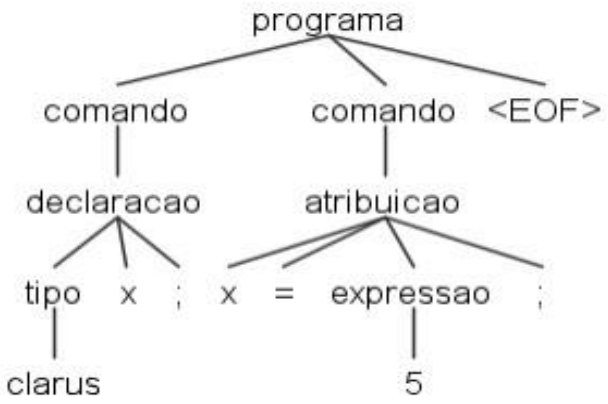
y; veritas z;




## Atribuição com tipos compatíveis

clarus x;  x

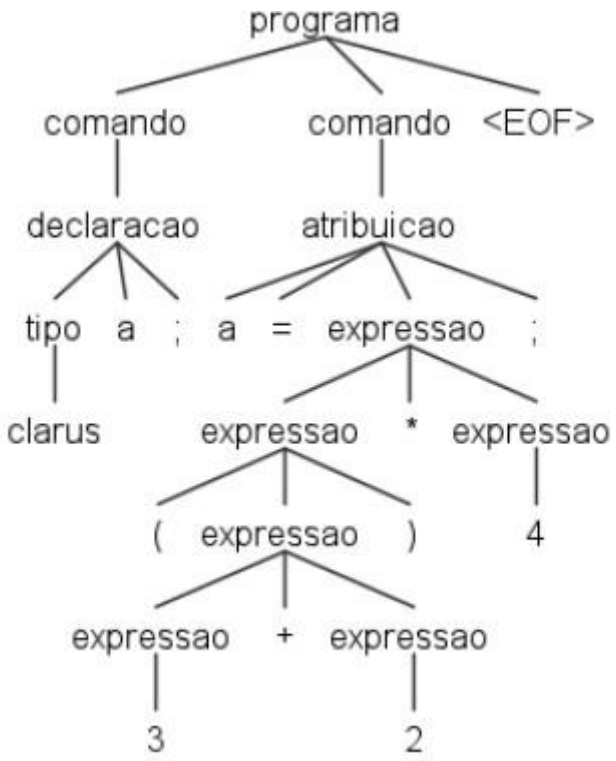
= 5;




## Expressões com precedência

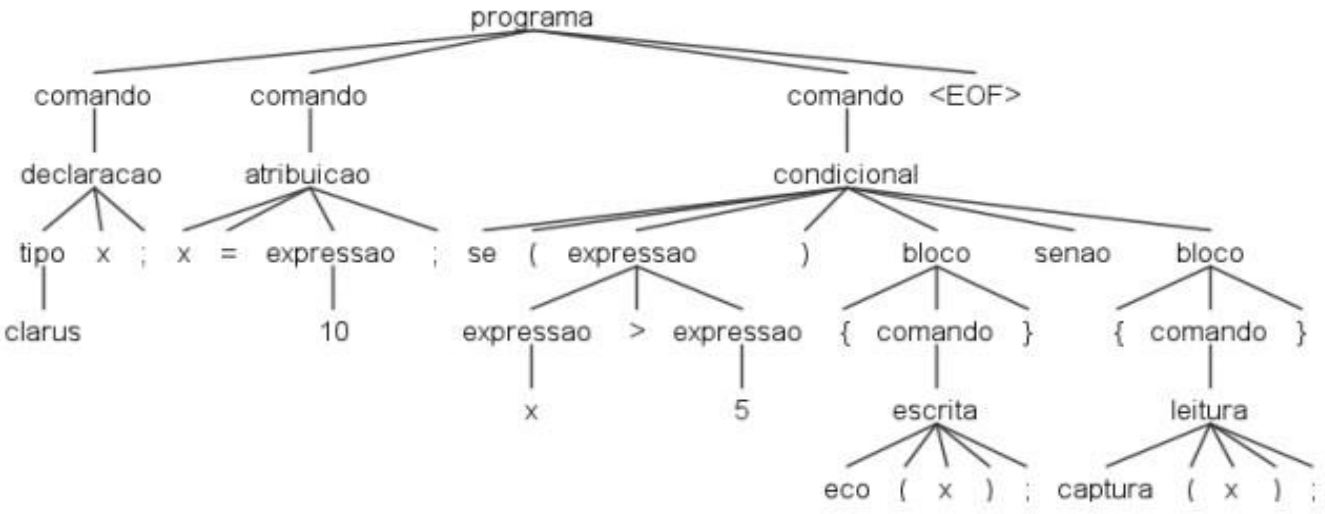
clarus a; a =

(3 + 2) * 4;



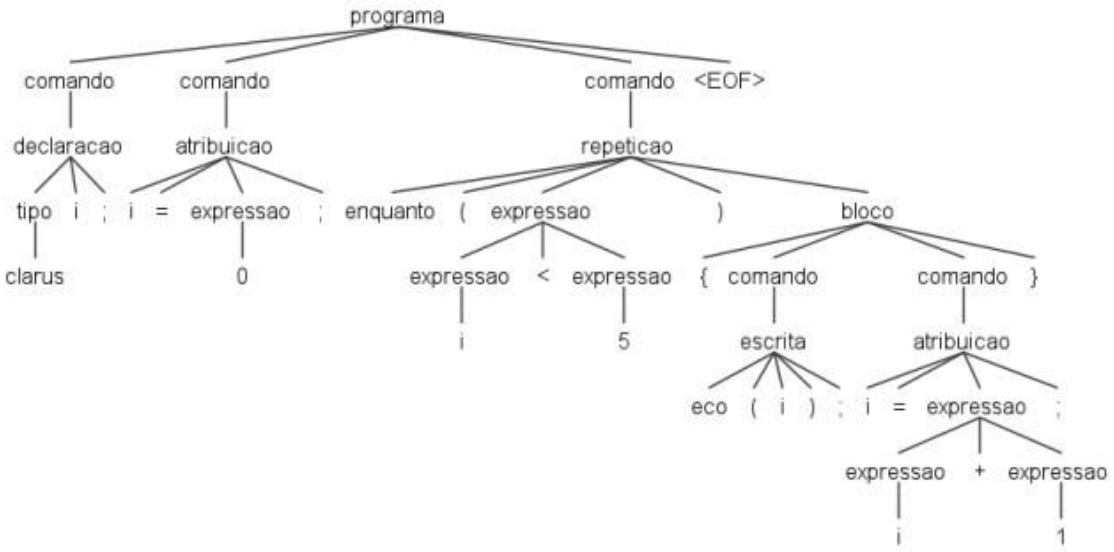

## Condicional com e sem senão

```
clarus x;   x =
10;  se (x > 5)
{       eco(x); }
senao           {
captura(x);
}
```




## Laço de repetição

```
clarus i;  i = 0;

enquanto (i < 5)

{

        eco(i);

i = i + 1;

                        }
```

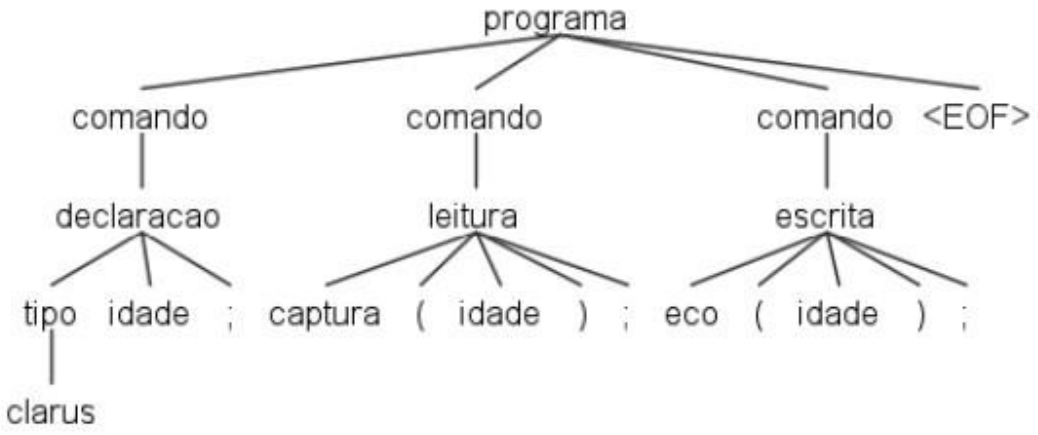



## Entrada e saída de dados

clarus idade;

captura(idade);

eco(idade);




As imagens foram geradas automaticamente pelo código **Teste.java** utilizando o componente TreeViewer da ANTLR e para cada exemplo **.lumin**, o arquivo foi salvo como **exemploX.lumin** e o código gerou **arvoreX.png**.

# Breve explicação de como a gramática trata cada requisito

## Tipos de variáveis

A regra **tipo** define três palavras-chave:

- **clarus**: números inteiros (**NUM_INT**)
- **flux**: números reais (**NUM_REAL**)
- **veritas**: valores booleanos (**verum, falsum**)

A regra **declaracao** permite declarar variáveis com esses tipos, e o analisador semântico (**LuminSemantico.java**) armazena cada variável em uma tabela de símbolos para checar tipo e declaração duplicada.

## Estrutura condicional (if...else)

A regra **condicional** define a estrutura:

**se (expressao) { bloco } senao { bloco }**

A expressão pode conter operadores relacionais como **==, !=, <, >**, definidos na regra **expressao**.

## Estrutura de repetição (while)

A regra **repeticao** permite a construção de laços do tipo **while**, utilizando:

**enquanto (expressao) { bloco }**

**A expressão é avaliada em tempo de execução, e seu tipo pode ser analisado no semântico.**

## Expressões aritméticas com precedência

A regra **expressao** usa a técnica de precedência implícita do ANTLR:

- Multiplicação e divisão são priorizadas com **ExprMulDiv**
- Soma e subtração vêm depois com **ExprSomaSub**
- Parênteses forçam agrupamento com **ExprParenteses**

## Atribuições com compatibilidade de tipos

A regra **atribuicao** define que variáveis recebem expressões:

**ID = expressao;**

No analisador semântico, é verificado se o tipo da variável e da expressão são compatíveis. Por exemplo:

- **clarus a = 5.7;** → ERRO (real em inteiro)
- **veritas b = 10;** → ERRO (inteiro em booleano)

## Verificação semântica (tipagem e escopo) O

analisador semântico **LuminSemantico.java**:

- Cria uma tabela de símbolos
- Verifica se variáveis foram declaradas antes de serem usadas
- Detecta redeclarações
- Compara os tipos esperados com os tipos das expressões atribuídas

## Entrada e saída

As regras **leitura** e **escrita** permitem entrada (**captura(...)**) e saída (**eco(...)**) de valores:

**captura(ID); eco(ID);**

## Espaços em branco e quebras de linha

A regra **WS** trata e ignora espaços, tabs e quebras de linha com:

**WS : [ \t\r\n]+ -> skip;**

Isso evita que esses caracteres atrapalhem a análise sintática.

## Compilador (EXTRA)

No projeto de compilador da linguagem Lumin, foi desenvolvido o método GeradorDeCodigo.java, responsável por transformar a árvore sintática gerada pelo ANTLR a partir da gramática original em um programa Java válido. Para isso, o gerador combina duas estratégias:

1. **Visitor –** percorre a árvore sintática de forma estruturada, visitando cada nó correspondente a comandos, expressões e declarações, e construindo gradualmente o código-fonte Java em memória.

2. **Listener –** captura eventos de entrada e saída de escopos específicos (por exemplo, início e fim de blocos if, while, declarações de variáveis etc.), para inserir automaticamente chaves, indentação e declarações auxiliares (imports, cabeçalhos de classe) no momento exato.

Ao final desse processamento, o método escreve todo o conteúdo gerado em um arquivo chamado saida.java, que contém o programa em Java correspondente ao código Lumin de entrada, com sintaxe, tipos e estruturas de controle devidamente traduzidos.

Essa abordagem híbrida (visitor + listener) garante tanto o controle fino sobre a geração de cada trecho de código quanto a correta organização do fluxo estrutural, resultando em um compilador simples e extensível.

---

## Drive com a gramática e Compilador: Lumin

---

## Referências:


https://vepo.medium.com/como-criar-uma-linguagem-usando-antlr4-ejava-ad834fadc2c1

https://www.tabnews.com.br/coffeeispower/antlr-o-jeito-mais-facil-de-criar-linguagens-de-programacao

https://github.com/antlr/grammars-v4