

# Análise Léxica e Sintática

**Beatriz Lopes Rizzo**

**João Pedro Basso Coura**

**João Pedro Figols Neco**

**Júlia Gachido Schmidt**

**Leonardo Fajardo Grupioni**

**PUC-SP**

**11 de junho, 2024**



**PUC-SP**

# Conteúdo

- Gramática
- Análise Sintática



# Gramática

funcao - **funcao** tipo nomeMetodo ( tipo variável );

funcao - **funcao** tipo nomeMetodo ( (tipo variável)\* ) { bloco **retorna** (número | variável) }

procedimento - **procedimento** nomeMetodo ( tipo variável );

procedimento - **procedimento** nomeMetodo ( (tipo variável)\* ) { bloco }

chamadaMetodo - variável ( variável );

declaração - tipo variável ;

declaração - tipo variável <- (**verdadeiro** | **falso**) ;

declaração - tipo variável <- termo ;

atribuiçãoVariável - chamadaMetodo;

atribuiçãoVariável - variável <- (**verdadeiro** | **falso**) ;

atribuiçãoVariável - variável <- termo ;

repetiçãoEnquanto - **enquanto** ( condição )

repetiçãoRepitaAte - **repita** { bloco } **ate que** ( condição )

repetiçãoRepitaPara - **para** variável **de** (variável | número) **ate** (variável | numero) **passo** número **faca**

instrução -> **se** (condição) **entao** { bloco }

instrução -> **se** (condição) **entao** { bloco } **senao** { bloco }

instrução -> **se** (condição) **entao** { bloco } **senao** **se** { bloco } **senao** { bloco }

condição -> expressao ( = | < | > | <= | >= ) expressao

expressão -> termo (( + | - | \* ) termo)\*

termo -> variável | numero ( . numero)\* | ( expressão )

bloco -> (declaração | chamadaMetodo | instrução | repetição)\*

variável -> **ID**

nomeMetodo -> **ID**

numero -> **NUM**

tipo -> **INT** | **DEC** | **CARAC** | **BOOL**



PUC-SP

# Análise Sintática

- procedimento - procedimento nomeMetodo ( tipo variável );
- procedimento - procedimento nomeMetodo ( (tipo variável)\* ) { bloco }

```
private void procedimento(){
    if(token.getNome() == nome.PROCEDIMENTO){
        no.acrescentarFilho(token.getNome() + ":" + token.getLexema());
        proxToken();

        if(token.getNome() == nome.PRINCIPAL){
            no.acrescentarFilho(token.getNome() + ":" + token.getLexema());
            proxToken();
            countPrinc++;
            Apar(no);
            Fpar(no);

            if(token.getNome() == nome.ACHAV){
                no.acrescentarFilho(token.getNome() + ":" + token.getLexema());
                proxToken();

                No bloco = no.acrescentarFilho("bloco");
                bloco(bloco);

                if(token.getNome() == nome.FCHAV){
                    no.acrescentarFilho(token.getNome() + ":" + token.getLexema());
                    proxToken();

                    Eof();
                } else erro(token);
            } else erro(token);
        }
    }
}
```

```
    } else {
        nomeMetodo(no);
        Apar(no);

        if(tipo()){
            No param = no.acrescentarFilho("parametro");
            tipo = false;
            param.acrescentarFilho(token.getNome() + ":" + token.getLexema());
            proxToken();

            if(variavel()){
                variavel = false;
                param.acrescentarFilho(token.getNome() + ":" + token.getLexema());
                proxToken();
            } else erro(token);
        }

        Fpar(no);

        if(token.getNome() == nome.PTV){
            no.acrescentarFilho(token.getNome() + ":" + token.getLexema());
            proxToken();

            procedimento();
        } else if(token.getNome() == nome.ACHAV){
            no.acrescentarFilho(token.getNome() + ":" + token.getLexema());
            proxToken();

            No bloco = no.acrescentarFilho("bloco");
            bloco(bloco);

            if(token.getNome() == nome.FCHAV){
                no.acrescentarFilho(token.getNome() + ":" + token.getLexema());
                proxToken();

                Eof();
            } else erro(token);
        } else erro(token);
    }

    } else funcao();
}
```



PUC-SP

# Análise Sintática

- funcao - funcao tipo nomeMetodo ( tipo variável );
- funcao - funcao tipo nomeMetodo ( (tipo variável)\* ) { bloco retorna (número | variável) }

```
private void funcao(){  
    if(token.getNome() == nome.FUNCAO){  
        no.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
  
        proxToken();  
  
        if(tipo()){  
            tipo = false;  
  
            no.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
  
            proxToken();  
  
            nomeMetodo(no);  
  
        }else erro(token);  
  
        Apar(no);  
  
        if(tipo()){  
            No param = no.acrescentarFilho("parametro");  
            tipo = false;  
            param.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(variavel()){  
                variavel = false;  
  
                param.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
  
                proxToken();  
            }else erro(token);  
        }  
  
        Fpar(no);  
    }  
}
```

```
if(token.getNome() == nome.PTV){  
    no.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
    proxToken();  
  
    procedimento();  
}  
else if(token.getNome() == nome.ACHAV){  
    no.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
    proxToken();  
  
    No bloco = no.acrescentarFilho("bloco");  
    bloco(bloco);  
  
    if(token.getNome() == nome.RETORNA){  
        bloco.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        if(token.getNome() == nome.NUM || token.getNome() == nome.ID){  
            bloco.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(token.getNome() == nome.PTV){  
                bloco.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                proxToken();  
  
                if(token.getNome() == nome.FCHAV){  
                    no.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                    proxToken();  
  
                    Eof();  
                }else erro(token);  
            }else erro(token);  
        }else erro(token);  
    }else erro(token);  
}
```



PUC-SP

# Análise Sintática

- bloco -> (declaração | chamadaMetodo | instrução | repetição)\*

```
private void bloco(No bloco){
    while(token.getNome() == nome.SE || token.getNome() == nome.ENQUANTO || token.getNome() == nome.REPITA || token.getNome() == nome.PARA
    || token.getNome() == nome.IMPRIMA || token.getNome() == nome.LEIA || token.getNome() == nome.ID || tipo()){
        if(token.getNome() == nome.SE){
            instrucao(bloco);
        }else if(token.getNome() == nome.ENQUANTO){
            repeticaoEnquanto(bloco);
        }else if(token.getNome() == nome.REPITA){
            repeticaoRepitaAte(bloco);
        }else if(token.getNome() == nome.PARA){
            repeticaoRepitaPara(bloco);
        }else if(token.getNome() == nome.IMPRIMA || token.getNome() == nome.LEIA){
            No comando = bloco.acrescentarFilho("comando");

            imprimaLeia(comando);
        }else if(token.getNome() == nome.ID){
            Token tokenAnt = token;

            proxToken();

            if(token.getNome() == nome.APAR){
                No chamada = bloco.acrescentarFilho("chamadaMetodo");
                No nomeMetodo = chamada.acrescentarFilho("nomeMetodo");
                nomeMetodo.acrescentarFilho(tokenAnt.getNome() + ":" + tokenAnt.getLexema());

                chamada.acrescentarFilho(token.getNome() + ":" + token.getLexema());
                proxToken();

                chamadaMetodo(chamada);
            }else{
                No atribuicao = bloco.acrescentarFilho("atribuicaoVariavel");
                No variavel = atribuicao.acrescentarFilho("variavel");
                variavel.acrescentarFilho(tokenAnt.getNome() + ":" + tokenAnt.getLexema());

                atribuicaoVariavel(atribuicao);
            }
        }else declaracao(bloco);
    }
}
```



PUC-SP

# Análise Sintática

```
private void Apar(No bloco){  
    if(token.getNome() == nome.APAR){  
        bloco.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    }else erro(token);  
}  
  
private void Fpar(No bloco){  
    if(token.getNome() == nome.FPAR){  
        bloco.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    }else erro(token);  
}  
  
private void Ptv(No bloco){  
    if(token.getNome() == nome.PTV){  
        bloco.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    }else erro(token);  
}
```



PUC-SP

# Análise Sintática

- atribuiçãoVariável - chamadaMetodo;
- atribuiçãoVariável - variável <- (verdadeiro | falso) ;
- atribuiçãoVariável - variável <- termo ;

```
private void atribuicaoVariavel(No atribuicao){  
    if(token.getNome() == nome.ATRIB){  
        atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        if(token.getNome() == nome.VERDADEIRO || token.getNome() == nome.FALSO){  
            atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(token.getNome() == nome.PTV){  
                atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                proxToken();  
            }else erro(token);  
        }else if(token.getNome() == nome.ARRAYC){  
            atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(token.getNome() == nome.PTV){  
                atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                proxToken();  
            }else erro(token);  
  
        }else if(token.getNome() == nome.ELCARAC){  
            atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(token.getNome() == nome.PTV){  
                atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                proxToken();  
            }else erro(token);  
        }  
    }  
}
```

```
}else if(token.getNome() == nome.ID){  
    Token tokenAnt = token;  
  
    proxToken();  
  
    if(token.getNome() == nome.APAR){  
        No nomeMetodo = atribuicao.acrescentarFilho("nomeMetodo");  
        nomeMetodo.acrescentarFilho(tokenAnt.getNome() + ":" + tokenAnt.getLexema());  
  
        atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        chamadaMetodo(atribuicao);  
    }else{  
        No variavel = atribuicao.acrescentarFilho("termo");  
        variavel.acrescentarFilho(tokenAnt.getNome() + ":" + tokenAnt.getLexema());  
  
        expressao(atribuicao);  
  
        if(token.getNome() == nome.PTV){  
            atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
        }else erro(token);  
    }  
}  
else{  
    expressao(atribuicao);  
  
    if(token.getNome() == nome.PTV){  
        atribuicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    }else erro(token);  
}  
}else erro(token);  
}
```



# Análise Sintática

- declaração - tipo variável ( <- ( (verdadeiro | falso) || termo )\*;

```
private void declaracao(No decl){  
    No declaracao = decl.acrescentarFilho("declaracao");  
  
    if(tipo()) {  
        tipo = false;  
        No tipo = declaracao.acrescentarFilho("tipo");  
        tipo.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    } else erro(token);  
  
    if(variavel()) {  
        variavel = false;  
        No variavel = declaracao.acrescentarFilho("variavel");  
        variavel.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    } else erro(token);  
  
    if(token.getNome() == nome.ACOL) {  
        declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        if(token.getNome() == nome.NUM) {  
            declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(token.getNome() == nome.FCOL) {  
                declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                proxToken();  
            } else erro(token);  
        } else erro(token);  
    }  
  
    if(token.getNome() == nome.PTV) {  
        declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
    } else if(token.getNome() == nome.ATRIB) {  
        declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        if(token.getNome() == nome.VERDADEIRO || token.getNome() == nome.FALSO) {  
            declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(token.getNome() == nome.PTV) {  
                declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                proxToken();  
            } else erro(token);  
        } else erro(token);  
    }  
}  
  
} else if(token.getNome() == nome.ARRAYC) {  
    declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
    proxToken();  
  
    if(token.getNome() == nome.PTV) {  
        declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    } else erro(token);  
} else if(token.getNome() == nome.ELCARAC) {  
    declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
    proxToken();  
  
    if(token.getNome() == nome.PTV) {  
        declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    } else erro(token);  
} else if(token.getNome() == nome.ID) {  
    Token tokenAnt = token;  
  
    proxToken();  
  
    if(token.getNome() == nome.APAR) {  
        No nomeMetodo = declaracao.acrescentarFilho("nomeMetodo");  
        nomeMetodo.acrescentarFilho(tokenAnt.getNome() + ":" + tokenAnt.getLexema());  
  
        declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        chamadaMetodo(declaracao);  
    } else {  
        No variavel = declaracao.acrescentarFilho("variavel");  
        variavel.acrescentarFilho(tokenAnt.getNome() + ":" + tokenAnt.getLexema());  
  
        expressao(declaracao);  
  
        if(token.getNome() == nome.PTV) {  
            declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
        } else erro(token);  
    }  
} else {  
    expressao(declaracao);  
  
    if(token.getNome() == nome.PTV) {  
        declaracao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    } else erro(token);  
}  
} else erro(token);
```



PUC-SP

# Análise Sintática

- condição -> expressao ( = | < | > | <= | >= ) expressao
- expressão -> termo (( + | - | \* ) termo)\*

```
private void condicao(No cond){  
    No condicao = cond.acrescentarFilho("condicao");  
    expressao(condicao);  
  
    if(token.getNome() == nome.IGUAL || token.getNome() == nome.MENOR || token.getNome() == nome.MAIOR || token.getNome() == nome.MAIORIGUAL || token.getNome() == nome.MENORIGUAL){  
        condicao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        expressao(condicao);  
    }  
}  
  
private void expressao(No exp){  
    No expressao = exp.acrescentarFilho("expressao");  
    termo(expressao);  
  
    while(token.getNome() == nome.MAIS || token.getNome() == nome.MENOS || token.getNome() == nome.MULT){  
        expressao.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        termo(expressao);  
    }  
}
```



PUC-SP

# Análise Sintática

- termo -> variável | numero (. numero)\* | ( expressão )

```
private void termo(No term){  
    if(token.getNome() == nome.ID){  
        No termo = term.acrescentarFilho("termo");  
        termo.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
    }else if(token.getNome() == nome.NUM){  
        No termo = term.acrescentarFilho("termo");  
        termo.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        if(token.getNome() == nome.PTO){  
            termo.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
  
            if(token.getNome() == nome.NUM){  
                termo.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
                proxToken();  
            }  
        }  
    }else if(token.getNome() == nome.APAR){  
        term.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
        proxToken();  
  
        expressao(term);  
  
        if(token.getNome() == nome.FPAR){  
            term.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
            proxToken();  
        }else erro(token);  
    }  
}
```



PUC-SP

# Análise Sintática

- chamadaMetodo - variável ( variável ) ;

```
private void chamadaMetodo(No chamada){  
    if(variavel()){  
        variavel = false;  
  
        No variavel = chamada.acrescentarFilho("variavel");  
        variavel.acrescentarFilho(token.getNome() + ":" + token.getLexema());  
  
        proxToken();  
    }  
  
    Fpar(chamada);  
    Ptv(chamada);  
}
```



PUC-SP

# Análise Sintática

- Árvore Sintática Abstrata

```
public class No
{
    private ArrayList<No> filhos;
    private String chave;

    public No(String chave){
        setChave(chave);
        filhos = new ArrayList<>();
    }

    private void setChave(String chave){
        this.chave = chave;
    }

    public No acrescentarFilho(String chave){
        No filho = new No(chave);
        filhos.add(filho);

        return filho;
    }

    public String getChave(){
        return this.chave;
    }
}
```

```
public class Arvore
{
    private No raiz;

    public Arvore(){
        raiz = new No("Inicio");
    }

    private void setRaiz(No raiz){
        this.raiz = raiz;
    }

    public No getRaiz(){
        return this.raiz;
    }
}
```



PUC-SP

# Análise Sintática

- Teste de um código simples de uma árvore

```
procedimento principal(){
    int t;
    int i <- 1;
    int k <- 2;
    int soma <- i + (k * 10);

    bool fim <- verdadeiro;
    soma <- ((9+10)*80)-10;

    dec l <- 1.9;
    nome();
}
```



PUC-SP

# Análise Sintática

```
Analise Sintatica Executada com Sucesso, sem detecção de erros!!!

raiz: Inicio
filhos: PROCEDIMENTO: procedimento PRINCIPAL: principal APAR: ( FPAR: ) ACHAV: { bloco FCHAV: }

raiz: bloco
filhos: declaracao declaracao declaracao declaracao declaracao atribuicaoVariavel declaracao chamadaMetodo

raiz: declaracao
filhos: tipo variavel PTV: ;

raiz: tipo
filhos: INT: int

raiz: variavel
filhos: ID: t

raiz: declaracao
filhos: tipo variavel ATRIB: <- expressao PTV: ;

raiz: tipo
filhos: INT: int

raiz: variavel
filhos: ID: i

raiz: expressao
filhos: termo

raiz: termo
filhos: NUM: 1

raiz: declaracao
filhos: tipo variavel ATRIB: <- expressao PTV: ;

raiz: tipo
filhos: INT: int

raiz: variavel
filhos: ID: k

raiz: expressao
filhos: termo
```



PUC-SP

# Análise Sintática

```
raiz: termo
filhos: NUM: 2

raiz: declaracao
filhos: tipo variavel ATRIB: <- termo expressao PTV: ;

raiz: tipo
filhos: INT: int

raiz: variavel
filhos: ID: soma

raiz: termo
filhos: ID: i

raiz: expressao
filhos: MAIS: + APAR: ( expressao FPAR: )

raiz: expressao
filhos: termo MULT: * termo

raiz: termo
filhos: ID: k

raiz: termo
filhos: NUM: 10

raiz: declaracao
filhos: tipo variavel ATRIB: <- VERDADEIRO: verdadeiro PTV: ;

raiz: tipo
filhos: BOOL: bool

raiz: variavel
filhos: ID: fim

raiz: atribuicaoVariavel
filhos: variavel ATRIB: <- expressao PTV: ;

raiz: variavel
filhos: ID: soma

raiz: expressao
filhos: APAR: ( expressao FPAR: ) MENOS: - termo
```

```
raiz: expressao
filhos: APAR: ( expressao FPAR: ) MULT: * termo

raiz: expressao
filhos: termo MAIS: + termo

raiz: termo
filhos: NUM: 9

raiz: termo
filhos: NUM: 10

raiz: termo
filhos: NUM: 80

raiz: termo
filhos: NUM: 10

raiz: declaracao
filhos: tipo variavel ATRIB: <- expressao PTV: ;

raiz: tipo
filhos: DEC: dec

raiz: variavel
filhos: ID: l

raiz: expressao
filhos: termo

raiz: termo
filhos: NUM: 1 PTO: . NUM: 9

raiz: chamadaMetodo
filhos: nomeMetodo APAR: ( FPAR: ) PTV: ;

raiz: nomeMetodo
filhos: ID: nome
```



PUC-SP

# Análise Sintática



# Análise Sintática

- Entrada e Saída

```
funcao int fatorial(int numero);

procedimento principal(){
    int resultado;
    int numero;
    leia(numero);
    resultado <- fatorial(numero);
    se (resultado = -1) entao {
        imprima("erro");
    } senao {
        imprima(resultado);
    }
}

funcao int fatorial(int numero){
    int resultado <- 1;
    int i;
    se (numero < 0) entao{
        resultado <- -1;
    } senao se (numero > 0) entao {
        para i de numero ate 1 passo -1 faca {
            resultado <- resultado * i;
        }
    }
    retorna resultado;
}
```



PUC-SP

# Análise Sintática

## • Entrada e Saída

```
[[@1, 0-5,"funcao", <FUNCAO>,1], [@2, 6-8,"int", <INT>,1], [@3, 9-16,"fatorial", <ID>,1], [@4, 17-17,"(", <APAR>,1], [@5, 18-20,"int", <INT>,1], [@6, 21-26,"numero", <ID>,1], [@7, 27-27,")", <FPAR>,1], [@8, 28-28,";", <PTV>,1], [@9, 29-40,"procedimento", <PROCEDIMENTO>,1], [@10, 41-49,"principal", <PRINCIPAL>,1], [@11, 50-50,"(", <APAR>,1], [@12, 51-51,")", <FPAR>,1], [@13, 52-52,"{", <ACHAV>,1], [@14, 53-55,"int", <INT>,1], [@15, 56-64,"resultado", <ID>,1], [@16, 65-65,";", <PTV>,1], [@17, 66-68,"int", <INT>,1], [@18, 69-74,"numero", <ID>,1], [@19, 75-75,";", <PTV>,1], [@20, 76-79,"leia", <LEIA>,1], [@21, 80-80,"(", <APAR>,1], [@22, 81-86,"numero", <ID>,1], [@23, 87-87,")", <FPAR>,1], [@24, 88-88,";", <PTV>,1], [@25, 89-97,"resultado", <ID>,1], [@26, 98-99,"-", <ATRIB>,1], [@27, 100-107,"fatorial", <ID>,1], [@28, 108-108,"(", <APAR>,1], [@29, 109-114,"numero", <ID>,1], [@30, 115-115,")", <FPAR>,1], [@31, 116-116,";", <PTV>,1], [@32, 117-118,"se", <SE>,1], [@33, 119-119,"(", <APAR>,1], [@34, 120-128,"resultado", <ID>,1], [@35, 129-129,"=", <IGUAL>,1], [@36, 130-130,"-", <MENOS>,1], [@37, 131-131,"1", <NUM>,1], [@38, 132-132,")", <FPAR>,1], [@39, 133-137,"entao", <ENTAO>,1], [@40, 138-138,"{", <ACHAV>,1], [@41, 139-145,"imprima", <IMPRIMA>,1], [@42, 146-146,"(", <APAR>,1], [@43, 147-152,"erro", <ARRAYC>,1], [@44, 153-153,")", <FPAR>,1], [@45, 154-154,";", <PTV>,1], [@46, 155-155,"}", <FCHAV>,1], [@47, 156-160,"senao", <SENAO>,1], [@48, 161-161,"{", <ACHAV>,1], [@49, 162-168,"imprima", <IMPRIMA>,1], [@50, 169-169,"(", <APAR>,1], [@51, 170-178,"resultado", <ID>,1], [@52, 179-179,")", <FPAR>,1], [@53, 180-180,";", <PTV>,1], [@54, 181-181,"}", <FCHAV>,1], [@55, 182-182,"}", <FCHAV>,1], [@56, 183-188,"funcao", <FUNCAO>,1], [@57, 189-191,"int", <INT>,1], [@58, 192-199,"fatorial", <ID>,1], [@59, 200-200,"(", <APAR>,1], [@60, 201-203,"int", <INT>,1], [@61, 204-209,"numero", <ID>,1], [@62, 210-210,")", <FPAR>,1], [@63, 211-211,"{", <ACHAV>,1], [@64, 212-214,"int", <INT>,1], [@65, 215-223,"resultado", <ID>,1], [@66, 224-225,"-", <ATRIB>,1], [@67, 226-226,"1", <NUM>,1], [@68, 227-227,";", <PTV>,1], [@69, 228-230,"int", <INT>,1], [@70, 231-231,"i", <ID>,1], [@71, 232-232,";", <PTV>,1], [@72, 233-234,"se", <SE>,1], [@73, 235-235,"(", <APAR>,1], [@74, 236-241,"numero", <ID>,1], [@75, 242-242,"<", <MENOR>,1], [@76, 243-243,"0", <NUM>,1], [@77, 244-244,")", <FPAR>,1], [@78, 245-249,"entao", <ENTAO>,1], [@79, 250-250,"{", <ACHAV>,1], [@80, 251-259,"resultado", <ID>,1], [@81, 260-261,"-", <ATRIB>,1], [@82, 262-262,"-", <MENOS>,1], [@83, 263-263,"1", <NUM>,1], [@84, 264-264,";", <PTV>,1], [@85, 265-265,"}", <FCHAV>,1], [@86, 266-270,"senao", <SENAO>,1], [@87, 271-272,"se", <SE>,1], [@88, 273-273,"(", <APAR>,1], [@89, 274-279,"numero", <ID>,1], [@90, 280-280,">", <MAIOR>,1], [@91, 281-281,"0", <NUM>,1], [@92, 282-282,")", <FPAR>,1], [@93, 283-287,"entao", <ENTAO>,1], [@94, 288-288,"{", <ACHAV>,1], [@95, 289-292,"para", <PARA>,1], [@96, 293-293,"i", <ID>,1], [@97, 294-295,"de", <DE>,1], [@98, 296-301,"numero", <ID>,1], [@99, 302-304,"ate", <ATE>,1], [@100, 305-305,"1", <NUM>,1], [@101, 306-310,"passo", <PASSO>,1], [@102, 311-311,"-", <MENOS>,1], [@103, 312-312,"1", <NUM>,1], [@104, 313-316,"faca", <FACA>,1], [@105, 317-317,"{", <ACHAV>,1], [@106, 318-326,"resultado", <ID>,1], [@107, 327-328,"-", <ATRIB>,1], [@108, 329-337,"resultado", <ID>,1], [@109, 338-338,"*", <MULT>,1], [@110, 339-339,"i", <ID>,1], [@111, 340-340,";", <PTV>,1], [@112, 341-341,"}", <FCHAV>,1], [@113, 342-342,"}", <FCHAV>,1], [@114, 343-349,"retorna", <RETORNA>,1], [@115, 350-358,"resultado", <ID>,1], [@116, 359-359,";", <PTV>,1], [@117, 360-360,"}", <FCHAV>,1], [@118, 361-361,"EOF", <EOF>,1]]
```

Analise Sintatica Executada com Sucesso, sem detecção de erros|



PUC-SP

# Análise Sintática

- Entrada e Saída

```
leia(numero);  
resultado <- fatorial(int numero);
```

Erro Sintático no Token: [@29, 109-111,"int", <INT>, 1]

