

1COP020 - Trabalho T1: Portugol

Portugol é uma pseudolinguagem que permite desenvolver algoritmos estruturados em português de forma relativamente mais simples e intuitiva, independentemente de linguagens de programação verdadeiras. Emprega-se a técnica de refinamentos sucessivos. Após o refinamento final, o algoritmo é codificado em alguma linguagem. A implementação de algoritmos desenvolvidos em Portugol é feita com facilidade a partir de um mapeamento para a linguagem de programação desejada. Ele existe desde a década de 1970, sendo utilizado para o aprendizado de linguagens de programação.

Os diagramas sintáticos ao fim do texto apresentam um dialeto de uma versão simplificada da linguagem Portugol. Nos diagramas, círculos, elipses e figuras com cantos arredondados correspondem a símbolos terminais da gramática, como por exemplo:

algoritmo variaveis inicio fim

entre outros. Retângulos e demais figuras com cantos em ângulos retos correspondem a símbolos não-terminais da gramática. Os símbolos **identificador**, **numero inteiro**, **numero real** e **string** são símbolos terminais, os quais são formados de acordo com as seguintes expressões regulares:

identificador	$[a-zA-Z_][a-zA-Z0-9_]^*$
numero inteiro	$[0-9]^+$
numero real	$[0-9]^+.[0-9]^+$

Os *tokens* **string**, são cadeias de caracteres entre aspas duplas. Considere o seguinte exemplo:

```
"Oi mundo, eu sou o Portugol!"
```

Nesta versão simplificado do Portugol, existem duas formas de comentário: comentário de bloco e comentário de linha. Os comentários podem ser feitos da seguinte forma:

```
{ texto do comentário de bloco }  
// texto do comentário de linha
```

Um fato importante sobre a linguagem Portugol, é que mesma não distingue entre letras maiúsculas e minúsculas. Desta forma, as variações

se SE Se sE

correspondem todas ao mesmo *token*, ou seja, a palavra reservada **se**. Da mesma forma, as variações

OI Oi oI oi

correspondem todas ao mesmo identificador.

Com base nesses diagramas, desenvolva um **analisador léxico** que **reconheça os tokens** dessa versão simplificada do Portugol. **Espaços em branco e quebras de linha devem ser descartados pelo analisador léxico sem causar erro.** Tabulações não precisam ser tratadas e não irão aparecer neste trabalho.

Também com base nos diagramas, desenvolva uma **gramática LL(1)** e implemente um **analisador sintático** para a sua gramática. O programa integrando os dois analisadores deve ser capaz de reconhecer erros léxicos e sintáticos.

Observação: O seu **analisador sintático** pode ser feito de duas maneiras a sua escolha:

- Analisador sintático **descendente recursivo**: cada não-terminal da gramática corresponde a uma função
- Analisador sintático **LL(1)**: implementado utilizando uma pilha para o reconhecimento

A linguagem Portugol deste trabalho possui o seguinte conjunto de palavras reservadas:

algoritmo	vetor	enquanto	imprima
inicio	matriz	faca	verdadeiro
fim	tipo	para	falso
variaveis	funcao	de	e
inteiro	procedimento	ate	ou
real	se	passo	nao
caractere	entao	repita	div
logico	senao	leia	

O seguinte conjunto de delimitadores também faz parte do Portugol deste texto:

Delimitador	Nome
;	ponto e vírgula
,	vírgula
:	dois pontos
.	ponto
[abre colchetes
]	fecha colchetes
(abre parênteses
)	fecha parênteses
=	igual
<>	diferente
>	maior
>=	maior igual
<	menor
<=	menor igual
+	mais
-	menos
*	vezes
/	divisão
<-	atribuição

Especificações do Trabalho

Sua implementação deve ser feita em C ou C++. A ferramenta **Flex NÃO** pode ser utilizada neste trabalho, você deve fazer manualmente o seu analisador léxico.

Em relação ao analisador sintático, você pode criar e utilizar uma gramática LL(1) em sua implementação ou ainda implementar um analisador descendente recursivo. A implementação do analisador sintático deve ser manual, isto é, nenhuma ferramenta de geração de *parsers* deve ser utilizada.

De forma a servir de ponto de partida, este texto contém também uma gramática que corresponde aos diagramas sintáticos apresentados. A gramática apresentada neste texto **não** é LL(1). Você pode utilizar ela como base para criar uma versão LL(1) da mesma, ou ainda criar a sua própria gramática do zero.

O seu programa deve ser gerado utilizando-se um **Makefile** e o executável gerado deve ter o nome de **portugol**.

O programa gerado deve ler as suas entradas da entrada padrão do sistema e imprimir as saídas na saída padrão do sistema. Um exemplo de execução para uma entrada chamada **teste.por** seria a seguinte:

```
$/portugol < teste.por
```

Se o programa que for fornecido como entrada estiver correto, a seguinte mensagem deve ser impressa:

```
PROGRAMA CORRETO.
```

Nenhuma linha extra deve ser gerada após a mensagem, ou seja, essa linha seria gerada pelo comando:

```
printf("PROGRAMA CORRETO.");
```

Erros léxicos devem ser indicados apresentando a linha e a coluna onde o mesmo ocorreu. Considere o seguinte código **portugol**:

```
Algoritmo teste;  
inicio  
    # <- 1;  
fim.
```

A saída gerada deve ser a seguinte:

```
ERRO LEXICO. Linha: 3 Coluna: 5 -> '#'
```

Observe que também deve ser impresso o *token* que não foi reconhecido pelo analisador léxico. Erros sintáticos devem apresentar a linha e a coluna onde o erro ocorreu. Considere o seguinte código **portugol**:

```
Algoritmo teste_2;  
inicio  
    1 <- 1;  
fim.
```

A saída gerada deve ser a seguinte:

```
ERRO DE SINTAXE. Linha: 3 Coluna: 5 -> '1'
```

Tanto para a mensagem de erro léxico ou sintático **não** deve haver quebra de linha extra, assim como ocorre na mensagem de programa correto. O seu programa só deve detectar e reportar o primeiro erro léxico ou sintático que encontrar (caso exista) no arquivo de entrada, e então finalizar o processo de análise desse arquivo.

Os arquivos de entrada **não** irão conter tabulações. Os tokens serão separados somente pelo caractere de espaço. Todos os comentários estarão completos, isto é, não haverá comentário iniciado e não terminado.

Recomendações

Por favor, evite escrever código da seguinte forma:

```
for(int i = 0; ...)
{
    ...
}
```

onde uma variável local está sendo declarada dentro de um comando. Se ainda sim quiser utilizar tal estilo de programação, não se esqueça de colocar no **Makefile** as devidas opções para que o compilador aceite tal construção, pois nem todos os compiladores a aceitam por padrão.

Em geral a opção `-std=c99` é o suficiente para que tal construção seja aceita e compilada sem maiores problemas. Sua utilização, em geral, é da seguinte forma:

```
$gcc -std=c99 teste.c -o teste
```

Se você programar utilizando C++ 11, por favor, utilize também a opção adequada do compilador para habilitar a compilação de tal versão do C++.

Especificações de Entrega

O trabalho deve ser entregue no AVA em um arquivo `.zip` com o nome **portugol.zip**. A entrega deve ser feita exclusivamente no AVA até a data/hora especificada. Não serão aceitas entregas atrasadas ou por outro meio que não seja o AVA.

Este arquivo `.zip` deve conter somente os arquivos necessários à compilação, sendo que deve haver um **Makefile** para a geração do executável.

Observação: o arquivo `.zip` **não** deve conter pastas, para que quando descompactado, os fontes do trabalho apareçam no mesmo diretório do `.zip`. O nome do executável gerado pelo **Makefile** deve ser **portugol**.

Os arquivos entregues serão compilados e testados da seguinte forma:

```
$unzip ./portugol.zip
$make
$./portugol < entrada.por > saida_aluno.txt
$diff saida_aluno.txt saida_esperada.txt
```

IMPORTANTE: Arquivos ou programas entregues fora do padrão receberão nota **ZERO**. Saídas geradas fora do padrão receberão nota **ZERO**. Entende-se como arquivo fora do padrão aquele que tenha um nome diferente de **portugol.zip** ou que contenha pastas. Entende-se como programa fora do padrão aquele que apresentar erro de compilação, que não ler da entrada padrão, não imprimir na saída padrão, por exemplo. Entende-se como saída fora do padrão aquela que quando comparada com a saída esperada utilizando-se **diff**, apresente diferenças com o arquivo de referência. Uma forma de verificar se seu arquivo, programa ou saída está dentro

das especificações é testar o mesmo com os comandos de compilação e teste apresentados no texto e comparar as saídas geradas com as saídas esperadas que são fornecidas no AVA. Se o seu arquivo/programa/saída **não** funcionar com o comandos, significa que ele está **fora** das especificações e, portanto, receberá nota **ZERO**.

IMPORTANTE: Se ficou com alguma dúvida em relação a qualquer item deste texto, não hesite em falar com o professor da disciplina, pois ele está à disposição para sanar eventuais dúvidas, além do que, isso faz parte do trabalho dele.

Gramática do Portugol que implementa os diagramas sintáticos

Programa → algoritmo identificador ; *BlocoVariaveis* *ProcedimentoFuncao* *BlocoComandos* .

ProcedimentoFuncao → *DeclaraProcedimento* *ProcedimentoFuncao*

ProcedimentoFuncao → *DeclaraFuncao* *ProcedimentoFuncao*

ProcedimentoFuncao →

DeclaraProcedimento → procedimento identificador *Parametros* ; *DeclaraParametros* *BlocoVariaveis*
BlocoComandos ;

DeclaraFuncao → funcao identificador *Parametros* : *TipoBasico* ; *DeclaraParametros* *BlocoVariaveis*
BlocoComandos ;

Parametros → (*DeclaraIdentificador*)

Parametros →

DeclaraParametros → *Declaracoes*

DeclaraParametros →

BlocoVariaveis → variaveis *Declaracoes*

BlocoVariaveis →

Declaracoes → *DeclaraVariaveis*

Declaracoes → *DeclaraTipo*

Declaracoes → *DeclaraVariaveis* *Declaracoes*

Declaracoes → *DeclaraTipo* *Declaracoes*

DeclaraTipo → tipo identificador = *VetorMatriz* [*Dimensao*] *TipoBasico* ;

DeclaraVariaveis → *TipoBasico* : *DeclaraIdentificador* ;

DeclaraIdentificador → identificador

DeclaraIdentificador → identificador , *DeclaraIdentificador*

VetorMatriz → vetor

VetorMatriz → matriz

Dimensao → numero_inteiro : numero_inteiro

Dimensao → numero_inteiro : numero_inteiro , *Dimensao*

TipoBasico → inteiro

TipoBasico → real

TipoBasico → caractere

TipoBasico → logico

TipoBasico → identificador

BlocoComandos → inicio *ListaComandos* fim

$ListaComandos \rightarrow Comandos ;$
 $ListaComandos \rightarrow Comandos ; ListaComandos$

$Comandos \rightarrow identificador$
 $Comandos \rightarrow identificador (ExprIter)$
 $Comandos \rightarrow Variavel \leftarrow Expressao$
 $Comandos \rightarrow se Expressao entao ListaComandos fim se$
 $Comandos \rightarrow se Expressao entao ListaComandos senao ListaComandos fim se$
 $Comandos \rightarrow enquanto Expressao faca ListaComandos fim enquanto$
 $Comandos \rightarrow para identificador de Expressao ate Expressao faca ListaComandos fim para$
 $Comandos \rightarrow para identificador de Expressao ate Expressao passo Expressao faca$
 $ListaComandos fim para$
 $Comandos \rightarrow repita ListaComandos ate Expressao$
 $Comandos \rightarrow leia (Variavel)$
 $Comandos \rightarrow imprima (ExprIter)$

$Expressao \rightarrow ExpressaoSimples$

$Expressao \rightarrow Expressao = ExpressaoSimples$
 $Expressao \rightarrow Expressao <> ExpressaoSimples$
 $Expressao \rightarrow Expressao < ExpressaoSimples$
 $Expressao \rightarrow Expressao \leq ExpressaoSimples$
 $Expressao \rightarrow Expressao \geq ExpressaoSimples$
 $Expressao \rightarrow Expressao > ExpressaoSimples$

$Expressao \rightarrow ExpressaoSimples EX$
 $EX \rightarrow = ExpressaoSimples EX$
 $EX \rightarrow <> ExpressaoSimples EX$
 $EX \rightarrow < ExpressaoSimples EX$
 $EX \rightarrow \leq ExpressaoSimples EX$
 $EX \rightarrow \geq ExpressaoSimples EX$
 $EX \rightarrow > ExpressaoSimples EX$
 $EX \rightarrow$

$ExpressaoSimples \rightarrow Termo$

$ExpressaoSimples \rightarrow + Termo$
 $ExpressaoSimples \rightarrow - Termo$
 $ExpressaoSimples \rightarrow ExpressaoSimples + Termo$
 $ExpressaoSimples \rightarrow ExpressaoSimples - Termo$
 $ExpressaoSimples \rightarrow ExpressaoSimples OU Termo$

$ExpressaoSimples \rightarrow Termo ES$
 $ExpressaoSimples \rightarrow + Termo ES$
 $ExpressaoSimples \rightarrow - Termo ES$
 $ES \rightarrow Termo ES$
 $ES \rightarrow + Termo ES$
 $ES \rightarrow - Termo ES$
 $ES \rightarrow$

$Termo \rightarrow Fator$

$Termo \rightarrow Termo * Fator$
 $Termo \rightarrow Termo / Fator$
 $Termo \rightarrow Termo DIV Fator$
 $Termo \rightarrow Termo E Fator$

$Termo \rightarrow Fator T$
 $T \rightarrow * Fator T$
 $T \rightarrow / Fator T$
 $T \rightarrow DIV Fator T$
 $T \rightarrow E Fator T$
 $T \rightarrow$

$Fator \rightarrow (Expressao)$

$Fator \rightarrow NAO Fator$

$Fator \rightarrow numero_inteiro$

$Fator \rightarrow numero_real$

$Fator \rightarrow verdadeiro$

$Fator \rightarrow falso$

$Fator \rightarrow string$

$Fator \rightarrow identificador (ExprIter)$

$Fator \rightarrow Variavel$

$Variavel \rightarrow \text{identificador}$

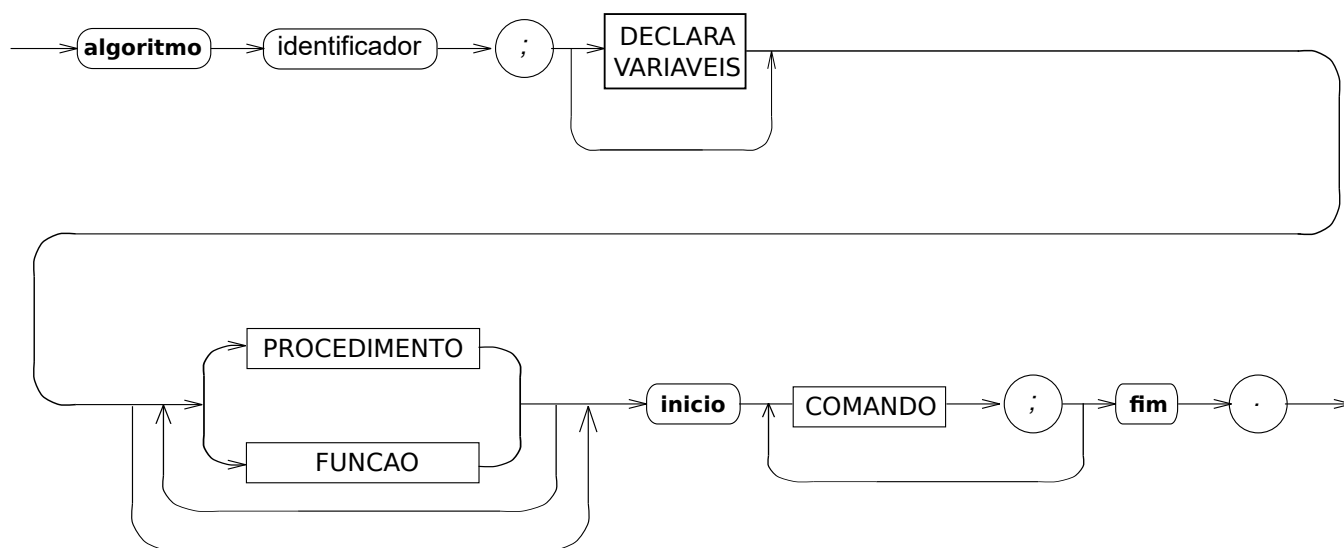
$Variavel \rightarrow \text{identificador} [ExprIter]$

$ExprIter \rightarrow Expressao$

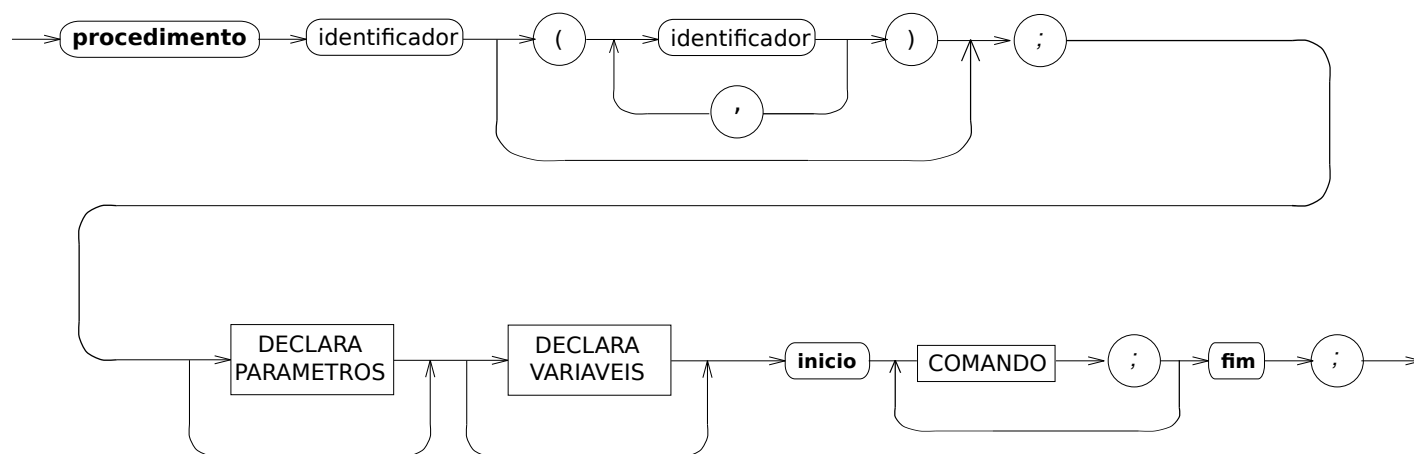
$ExprIter \rightarrow Expressao , ExprIter$

Portugol: Diagramas Sintáticos

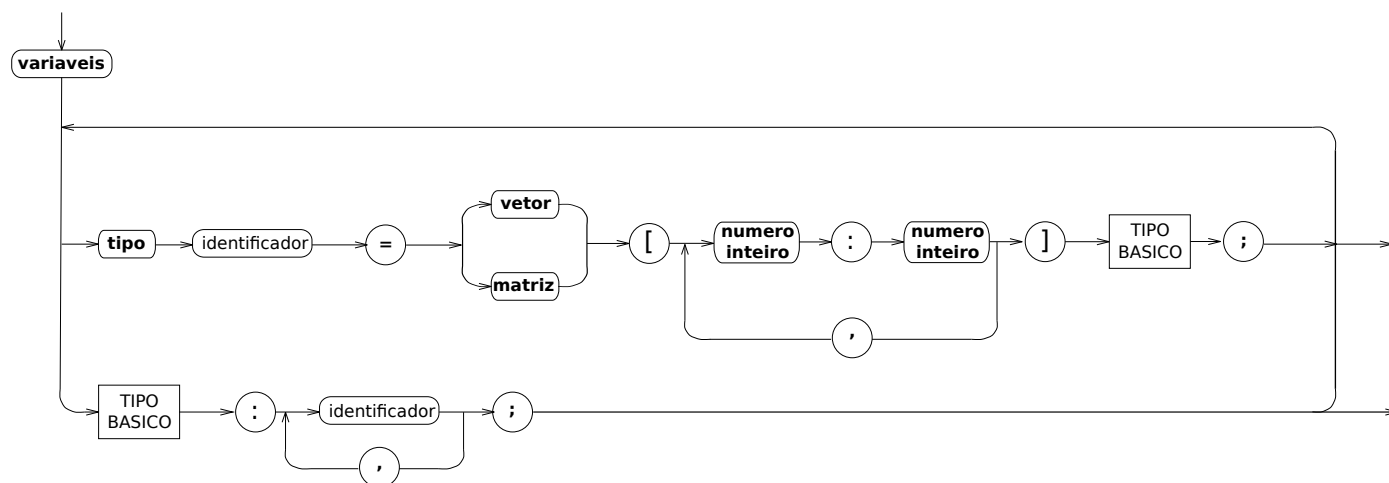
Portugol:



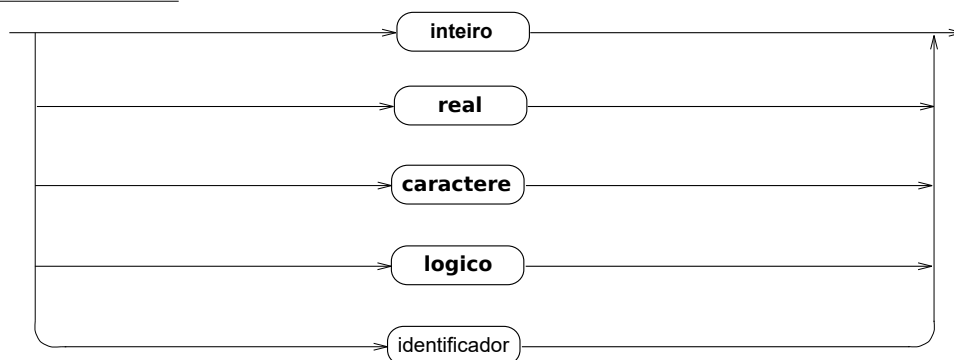
PROCEDIMENTO:



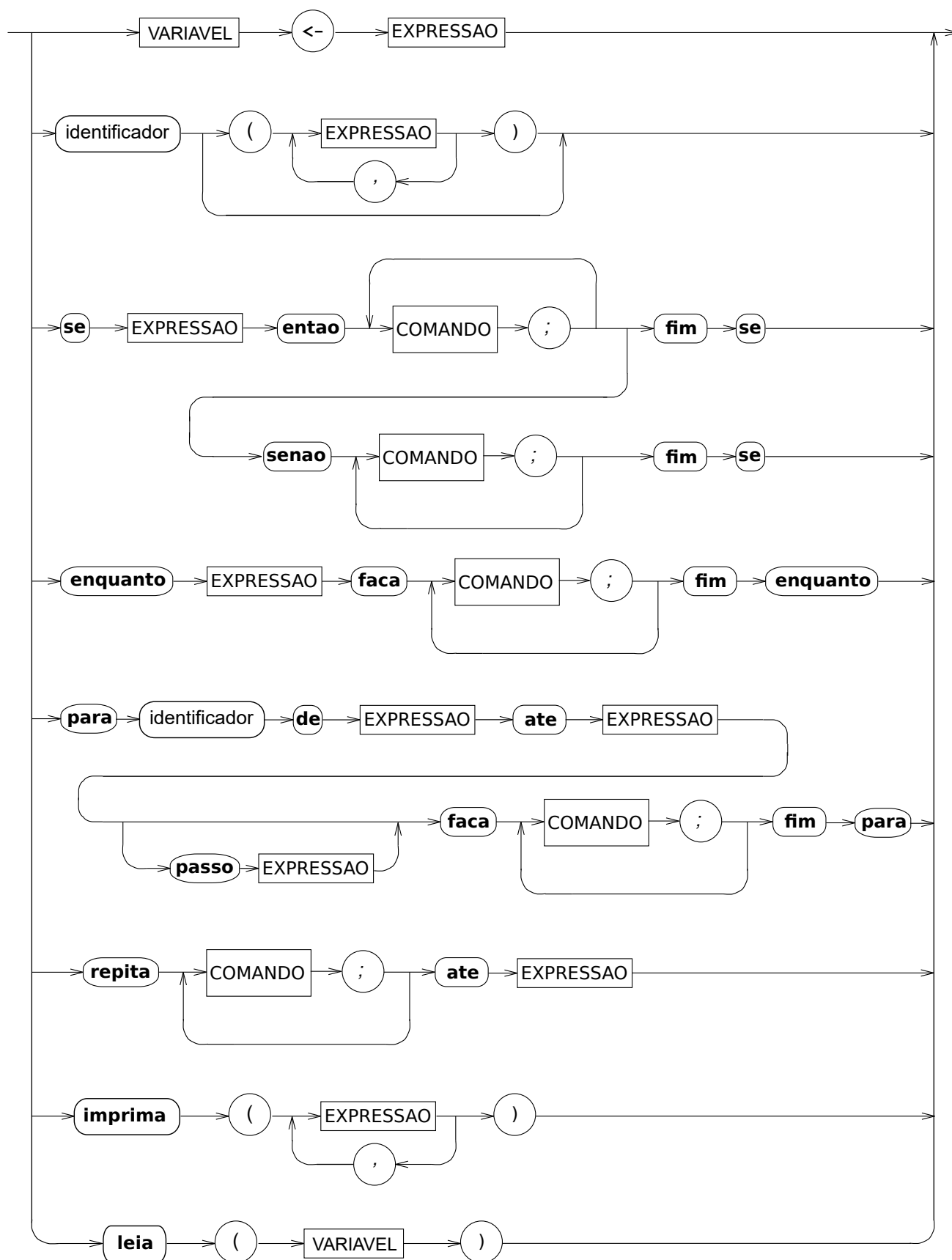
DECLARA VARIÁVEIS:



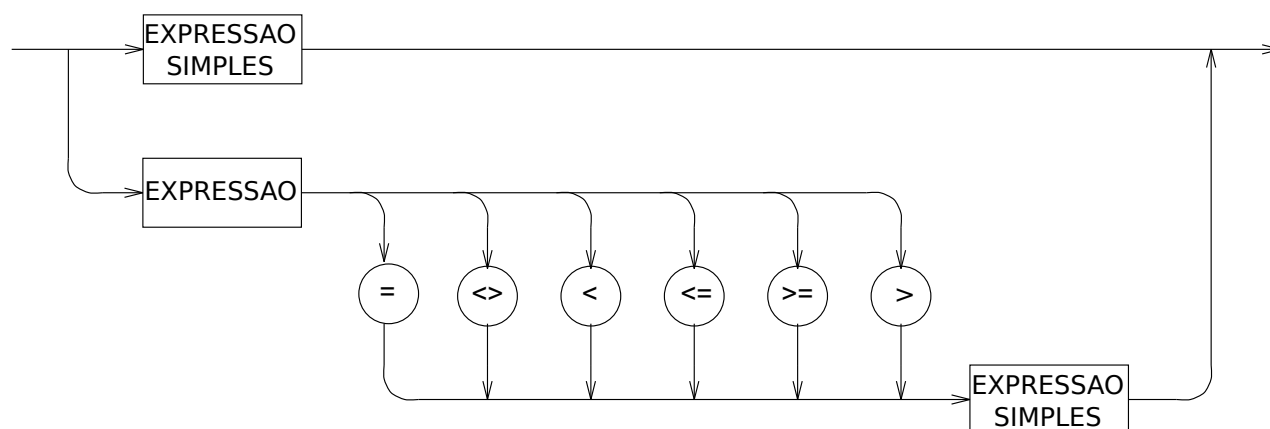
TIPO BASICO:



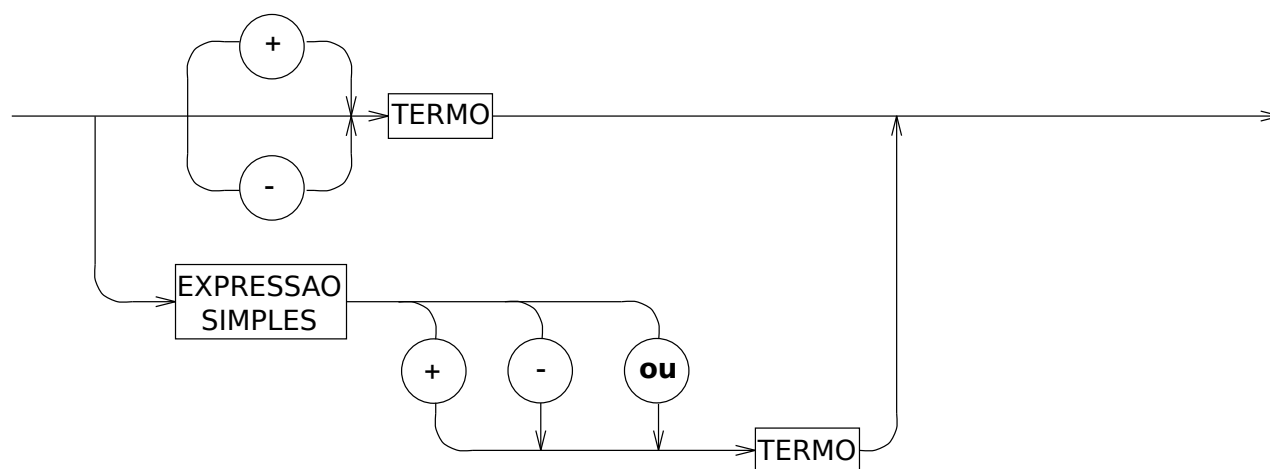
COMANDO:



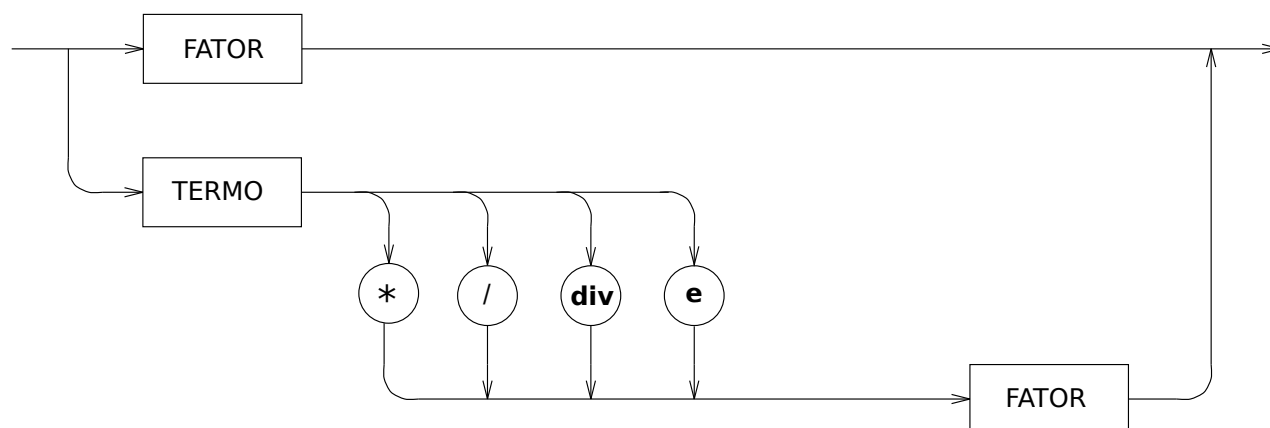
EXPRESSAO:



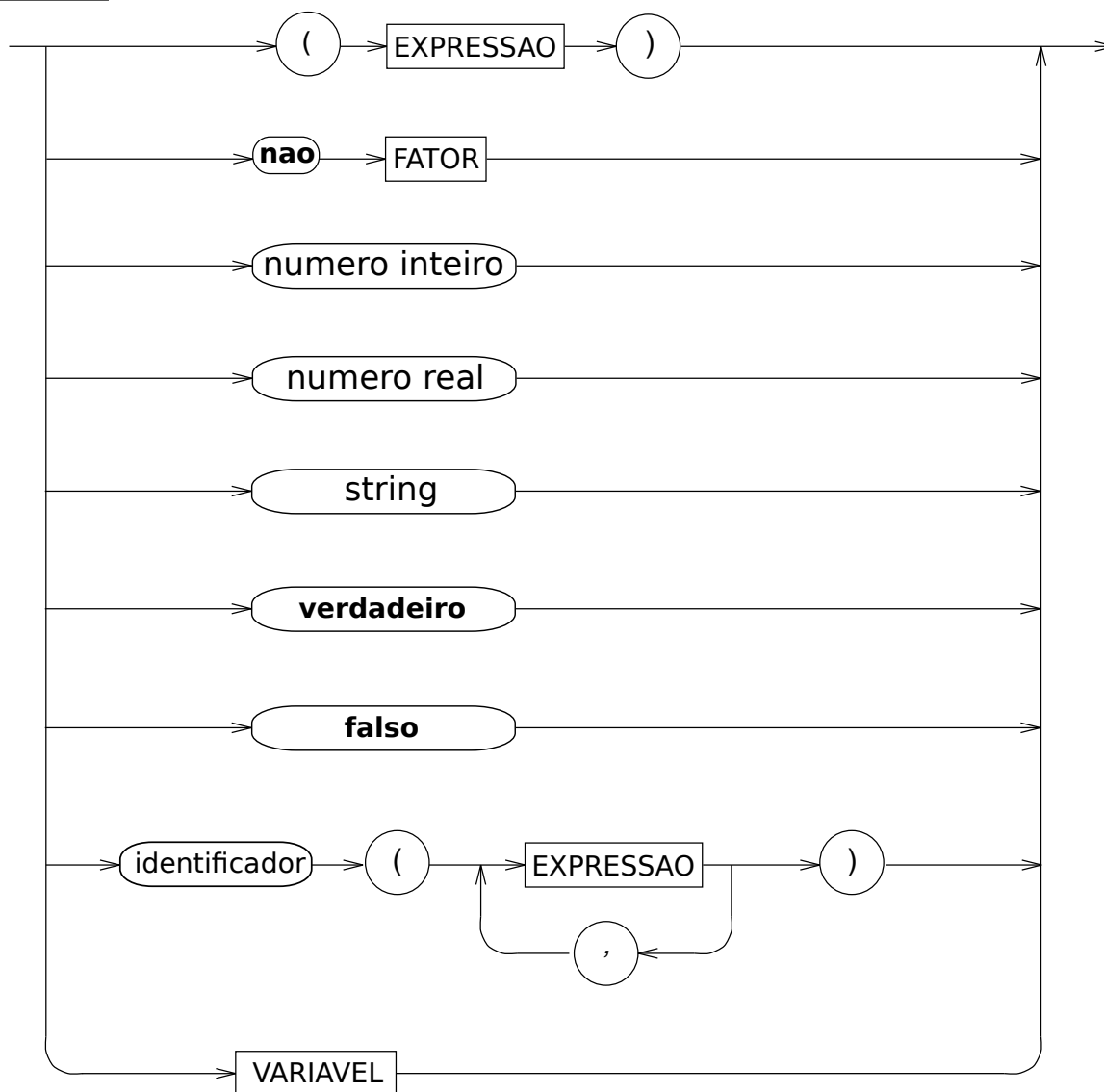
EXPRESSAO SIMPLES:



TERMO:



FATOR:



VARIAVEL:

