Universidade Estadual de Londrina

DC Departamento de Computação

5COP093 - Trabalho T1

A linguagem de programação Pascal foi criada em 1970 por Niklaus Wirth. Ela é uma linguagem estruturada, sendo muitas vezes utilizada para ensinar programação.

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

program var procedure begin

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

identificador	`[a-zA-Z_][a-zA-Z0-9_]`*
número	`[0-9]`$^+$

Com base nesses diagramas, desenvolva um analisador léxico que reconheça os *tokens* dessa versão simplificada do Pascal. Em seguida, também com base nos diagramas, desenvolva uma **gramática LL(1)** e implemente um analisador sintático LL(1) para a sua gramática. O programa integrando os dois analisadores deve ser capaz de reconhecer erros léxicos e sintáticos.

O seu analisador sintático LL(1) pode ser implementado utilizando uma pilha para o reconhecimento ou utilizar a técnica de análise descendente recursiva.

Em relação ao analisador léxico, o mesmo também deve ser capaz de remover comentários. Em Pascal, existe somente o conceito de comentário de bloco. No Pascal os comentários em bloco podem ser feitos de duas formas:

{	texto do comentário	}
`(*`	texto do comentário	`*)`

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.

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

var VAR Var vAr vAR vaR VAr VaR

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

OI Oi oI oi

correspondem todas ao mesmo identificador.

Especificações do Trabalho

Sua implementação deve ser feita em C ou C++. A ferramenta `Flex` ****não**** pode ser utilizada neste trabalho. Você deve implementar manualmente o analisador léxico. Em relação ao analisador sintático LL(1), a sua implementação deve ser manual; nenhuma ferramenta de geração de *parsers* deve ser utilizada.

O programa deve ser gerado utilizando-se um `Makefile` e o executável gerado deve ter o nome de `pascal`.

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.pas` seria a seguinte:

```
$./pascal < teste.pas
```

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 pascal:

```
program teste;
begin
    # := 1;
end.
```

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 onde o erro ocorreu. Considere o seguinte código pascal:

```
program teste;
begin
    1 := 1;
end.
```

A saída gerada deve ser a seguinte:

```
ERRO DE SINTAXE. Linha: 3 -> "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 sintátio que encontrar (caso exista) no arquivo de entrada, e então finalizar o processo de análise desse arquivo.

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 `pascal.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 `pascal`.

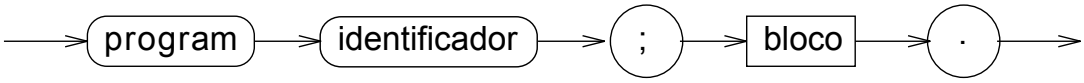
Os arquivos entregues serão compilados e testados da seguinte forma:

```
$unzip ./pascal.zip
$make
$./pascal < entrada.pas > saida_aluno.txt
$diff saida_aluno.txt saida_esperada.txt
```

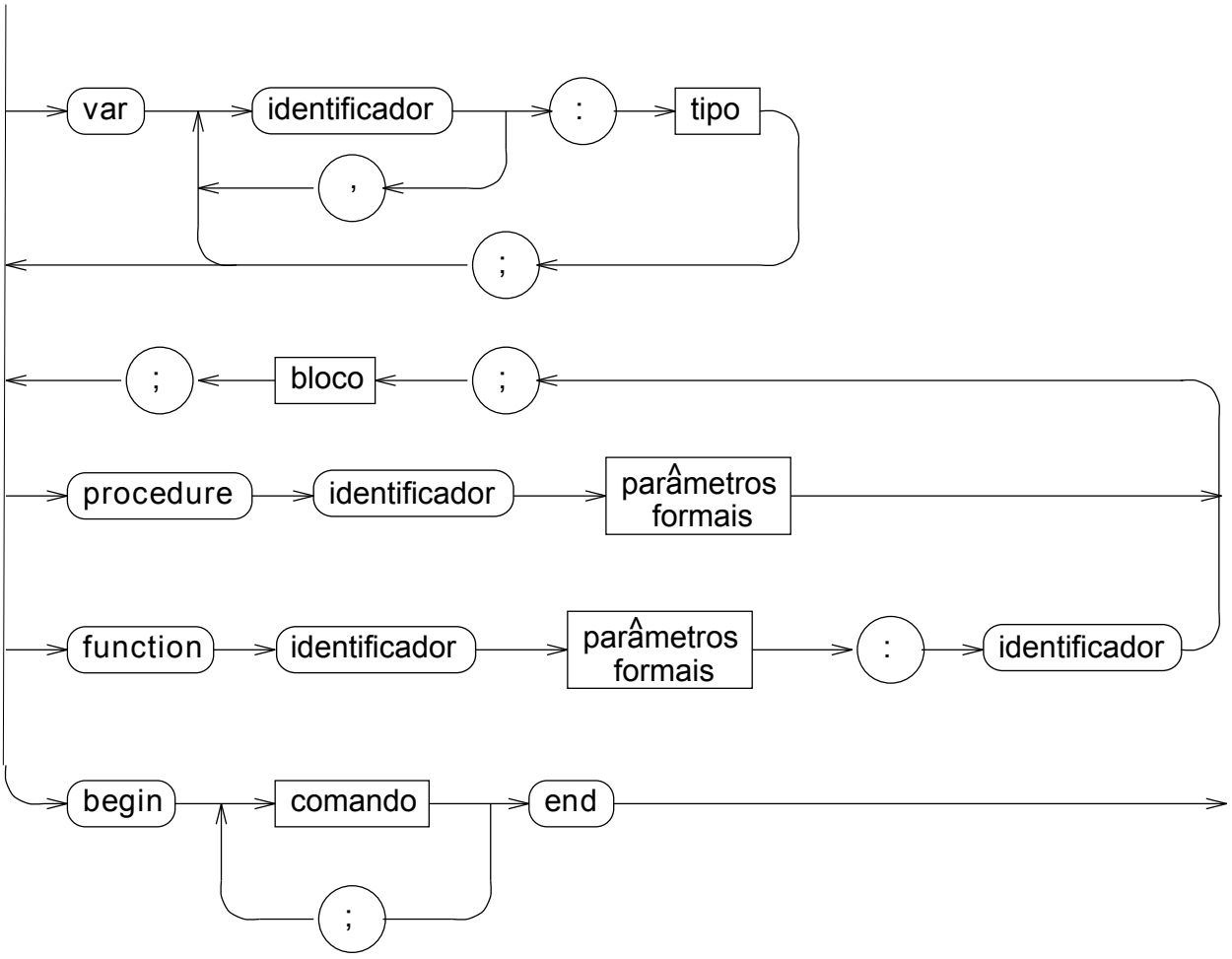
IMPORTANTE: Arquivos ou programas entregues fora do padrão receberão nota **ZERO**. Saídas geradas fora do padrão receberão nota **ZERO**. Entende-se como arquivo fora do padrão aquele que tenha um nome diferente de `pascal.zip` ou que contenha pastas. Entende-se como programa fora do padrão aquele que 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.

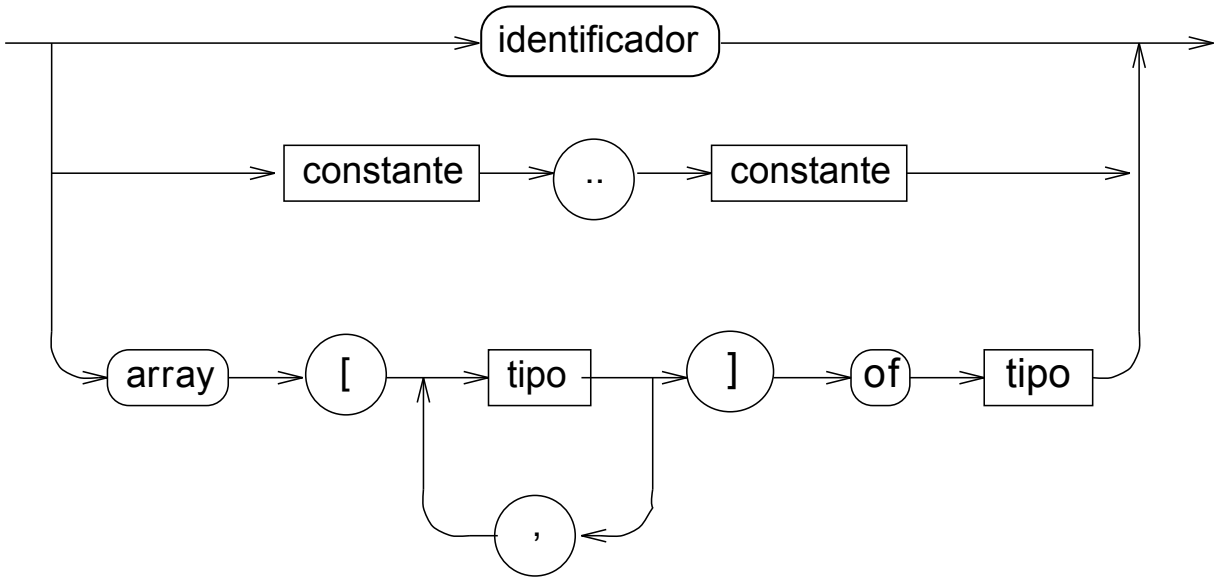
programa:



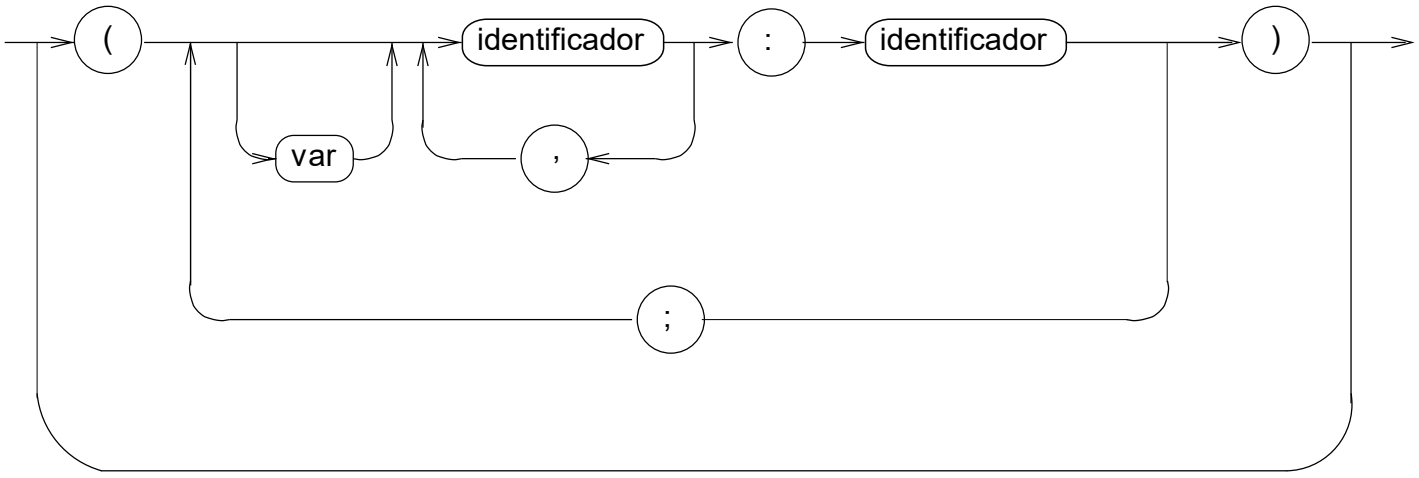
bloco:



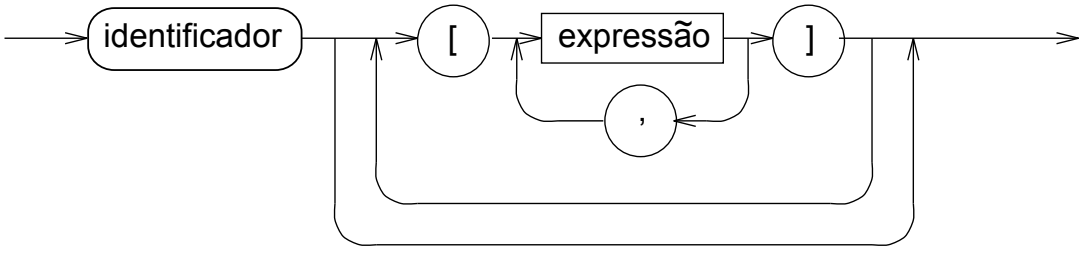
tipo:



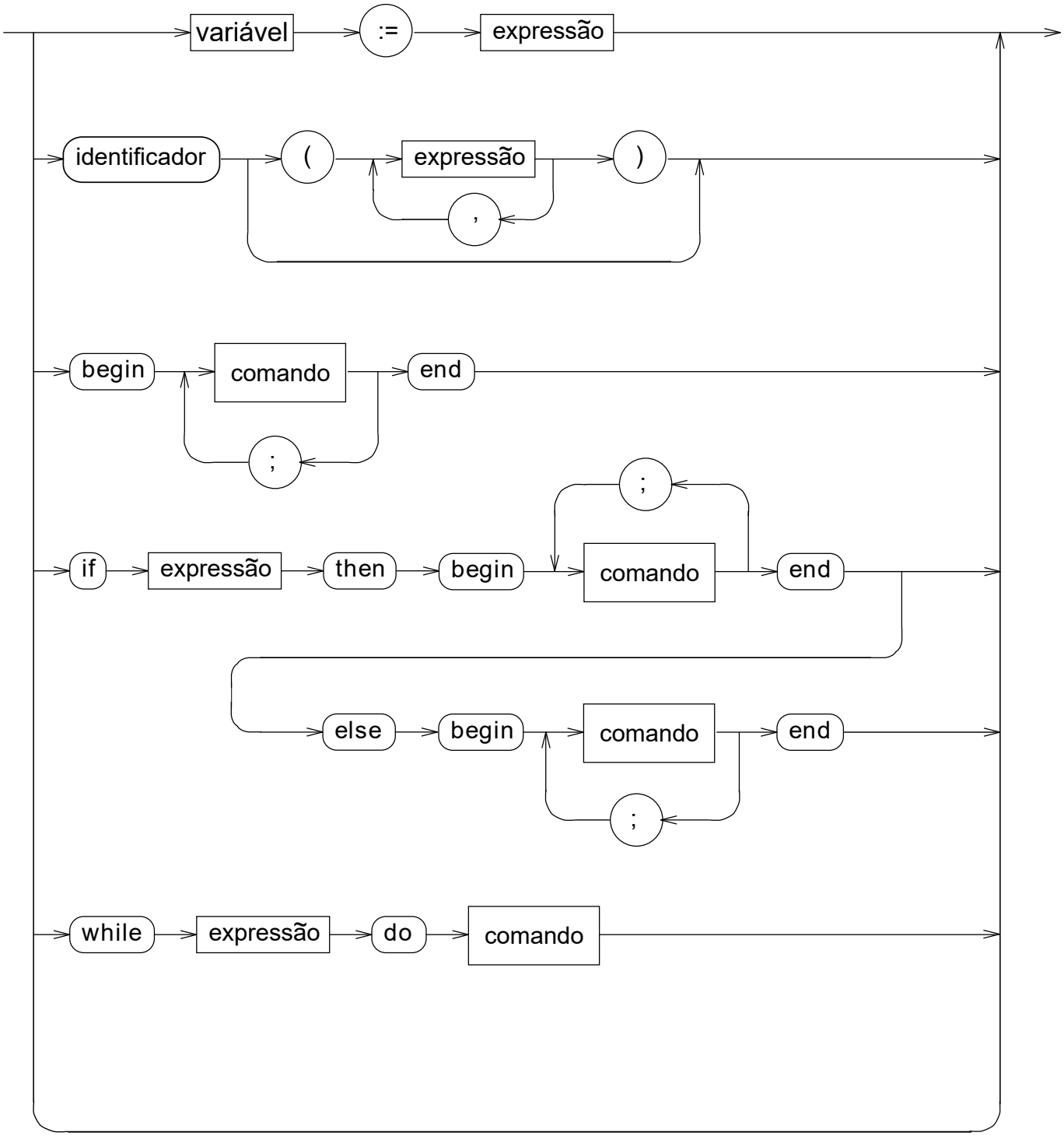
parâmetros formais:



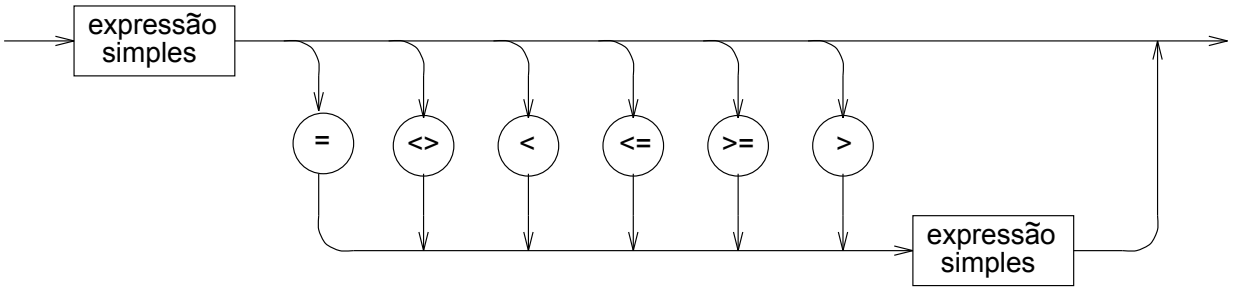
variável:



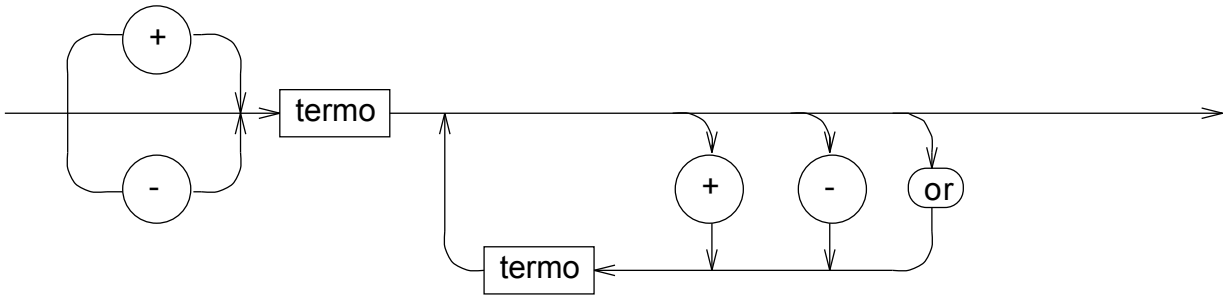
comando:


variável
:=
expressão
identificador
(
expressão
,
)
begin
comando
;
end
if
expressão
then
begin
comando
;
end
else
begin
comando
;
end
while
expressão
do
comando

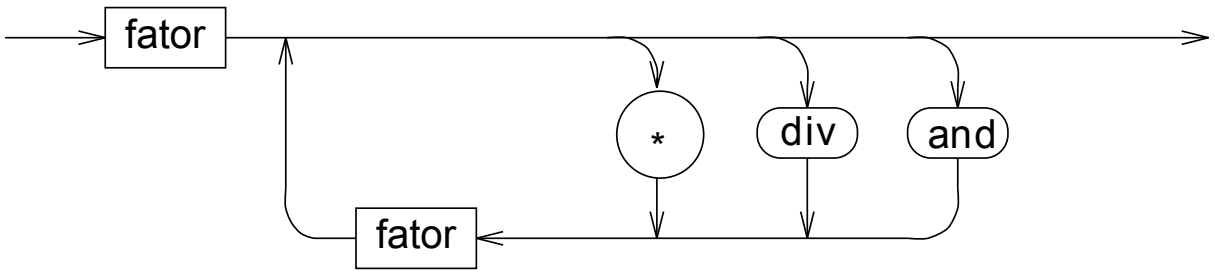
expressão:



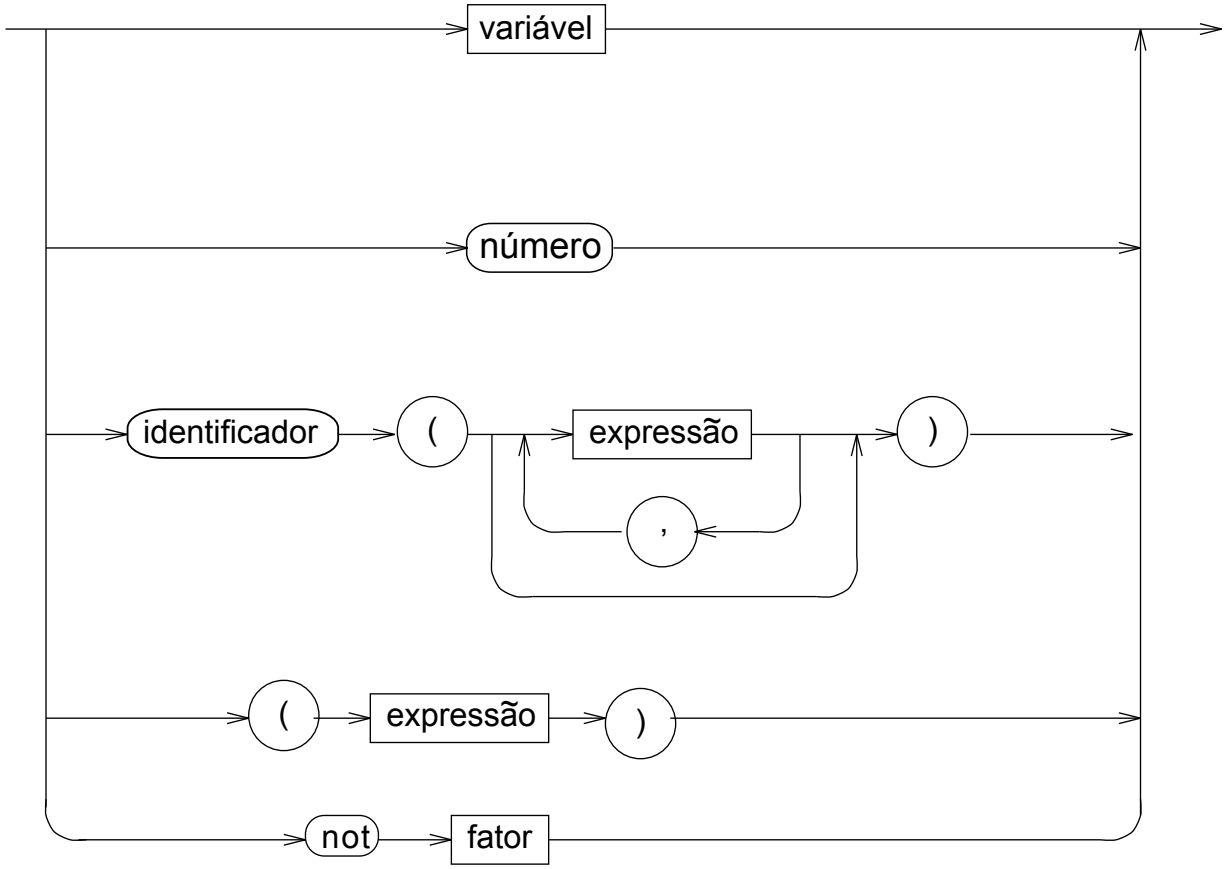
expressão simples:



termo:



fator:



constante: