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.

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, tabulações e quebras de linha devem ser descartados pelo analisador léxico sem causar erro.

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 de Entrega

O trabalho deve ser entregue no *moodle* em um arquivo .zip com o nome `pascal.zip`. Este arquivo .zip deve conter somente os arquivos necessários à compilação, sendo que deve have uma `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`.

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:

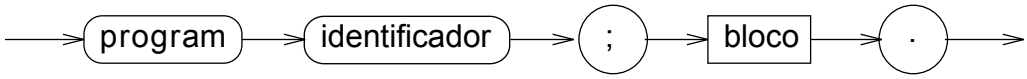
```
program teste;
begin
    1 := 1;
end.
```

A saída gerada deve ser a seguinte:

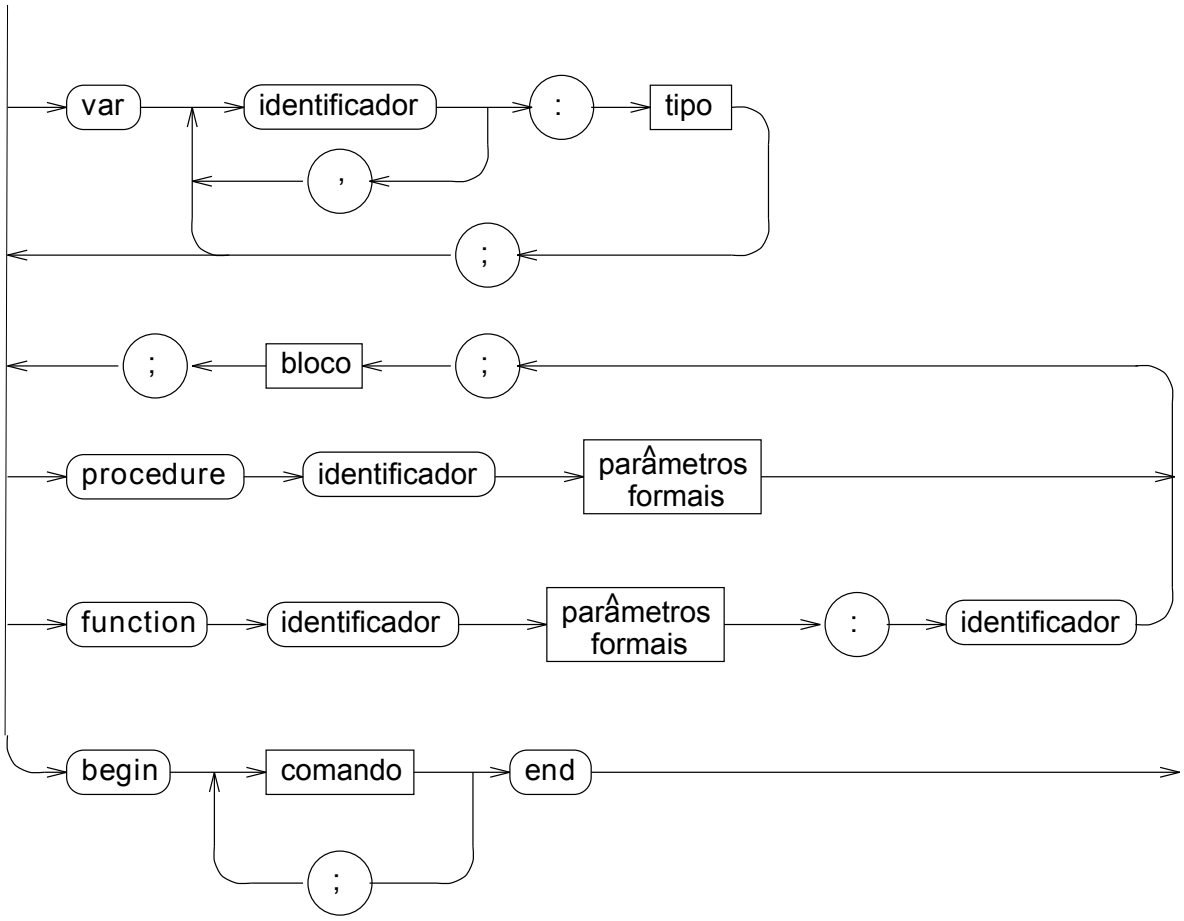
```
ERRO DE SINTAXE. Linha: 3 -> "1"
```

Tanto para a mensagem de erro léxico ou sintático **não** deve haver quebra de linha extra, assim como ocorre na mensagem de programa correto.

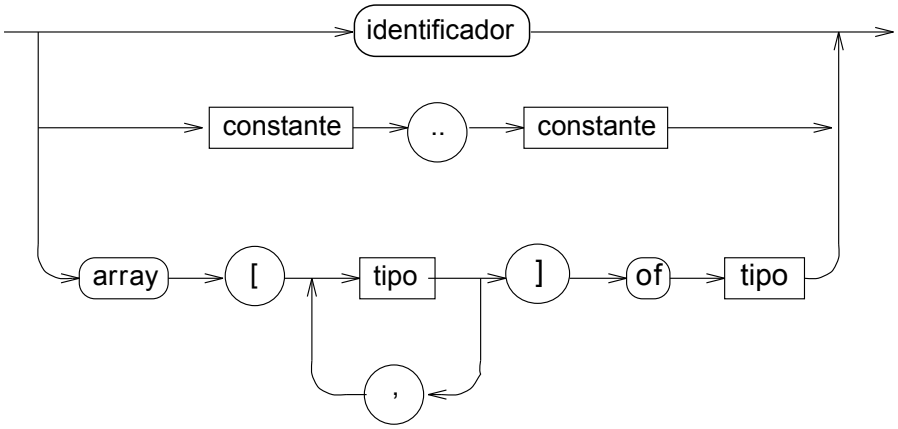
programa:



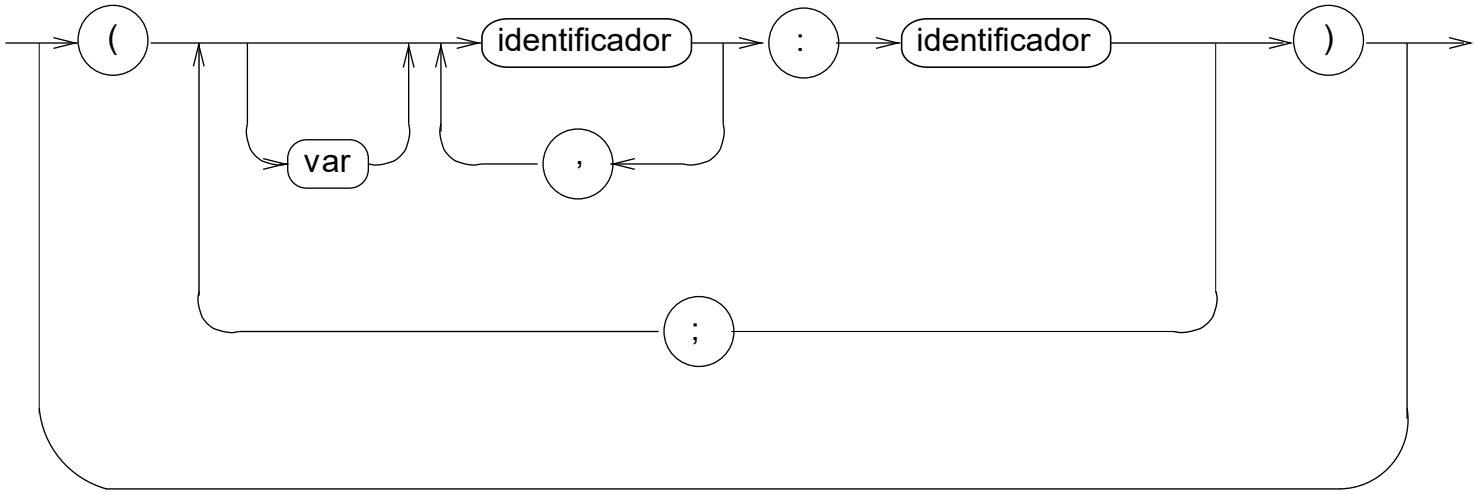
bloco:



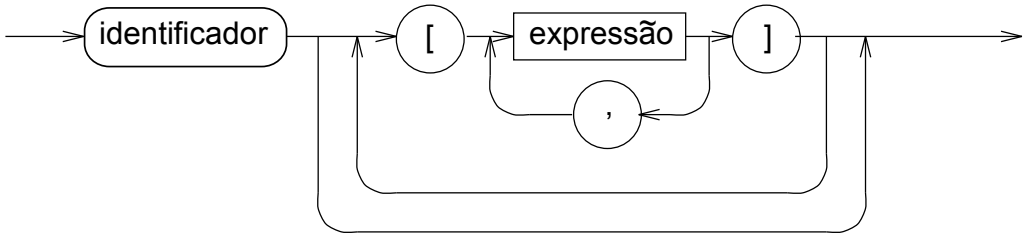
tipo:



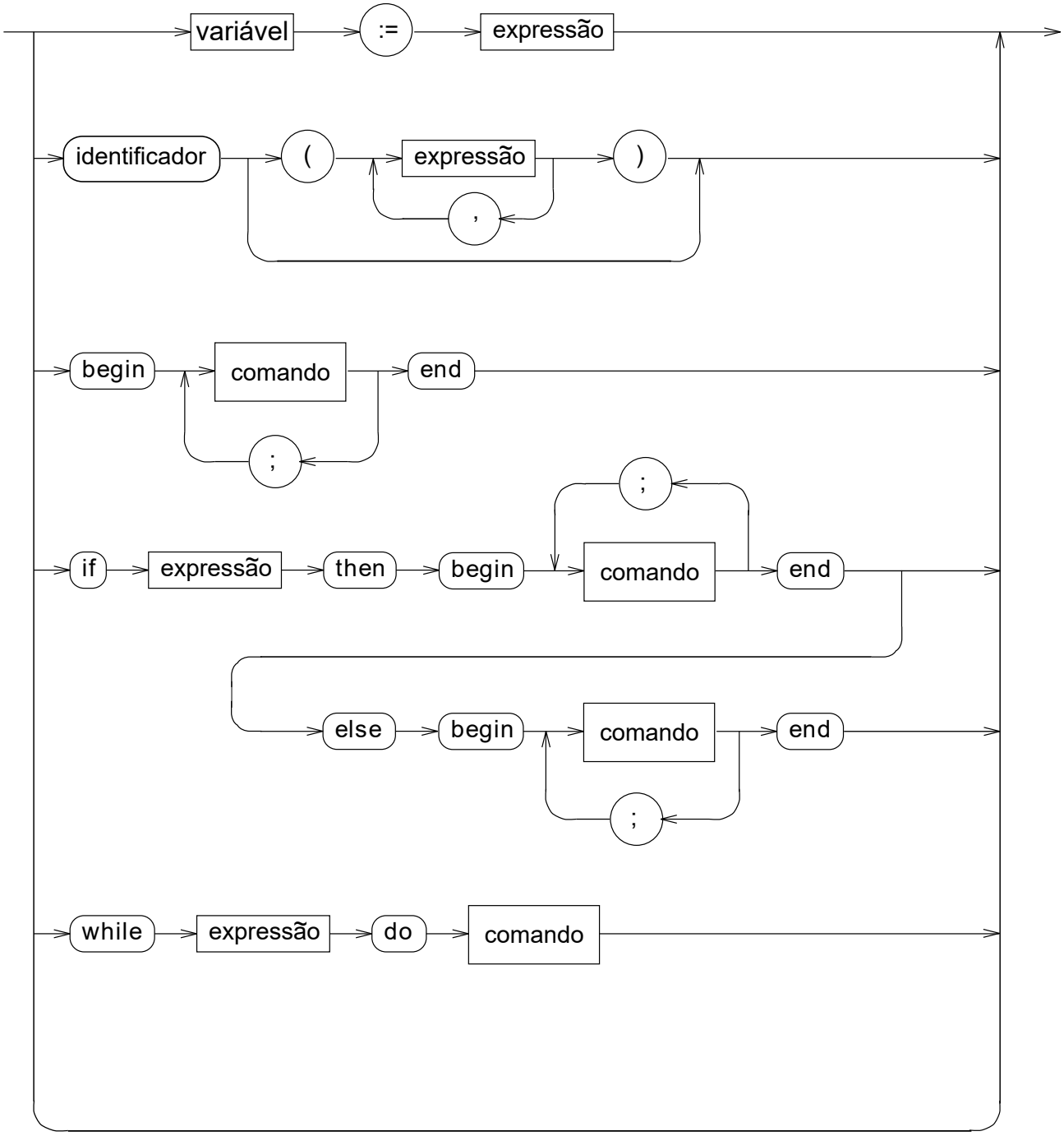
parâmetros formais:



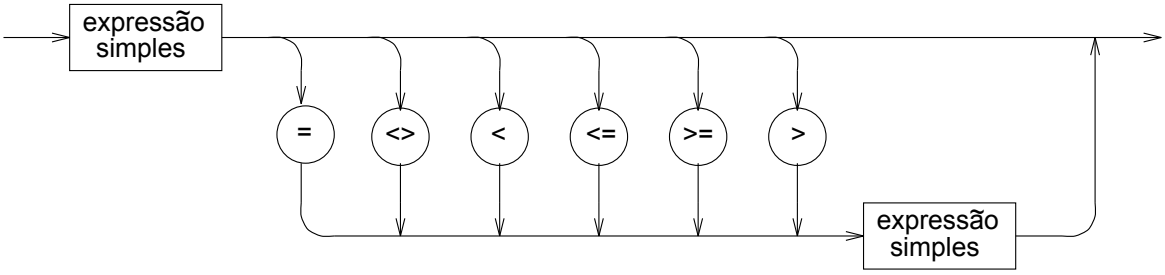
variável:



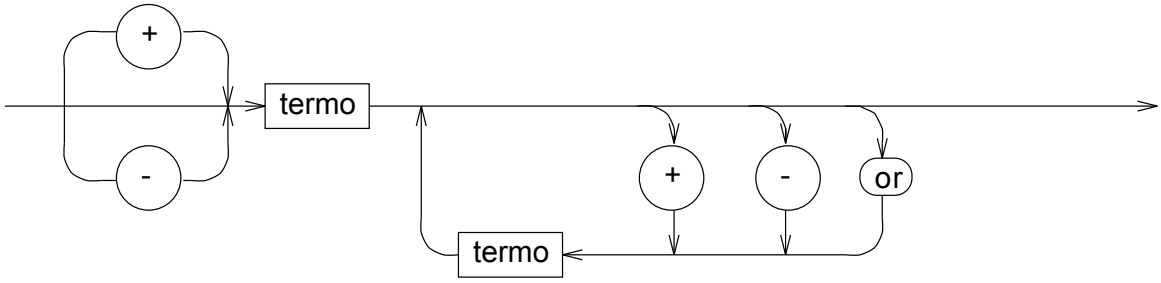
comando:


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

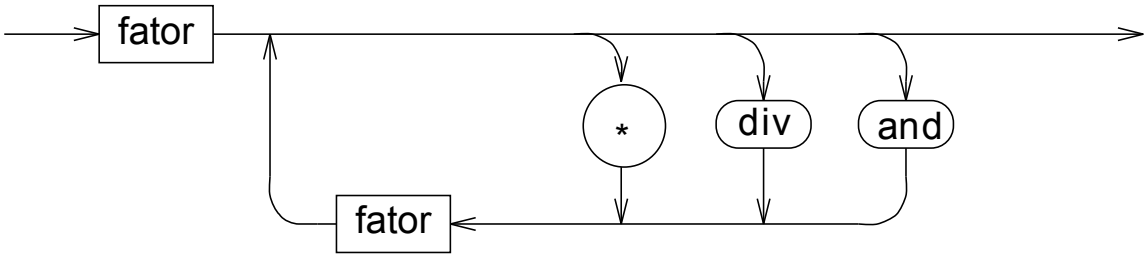
expressão:



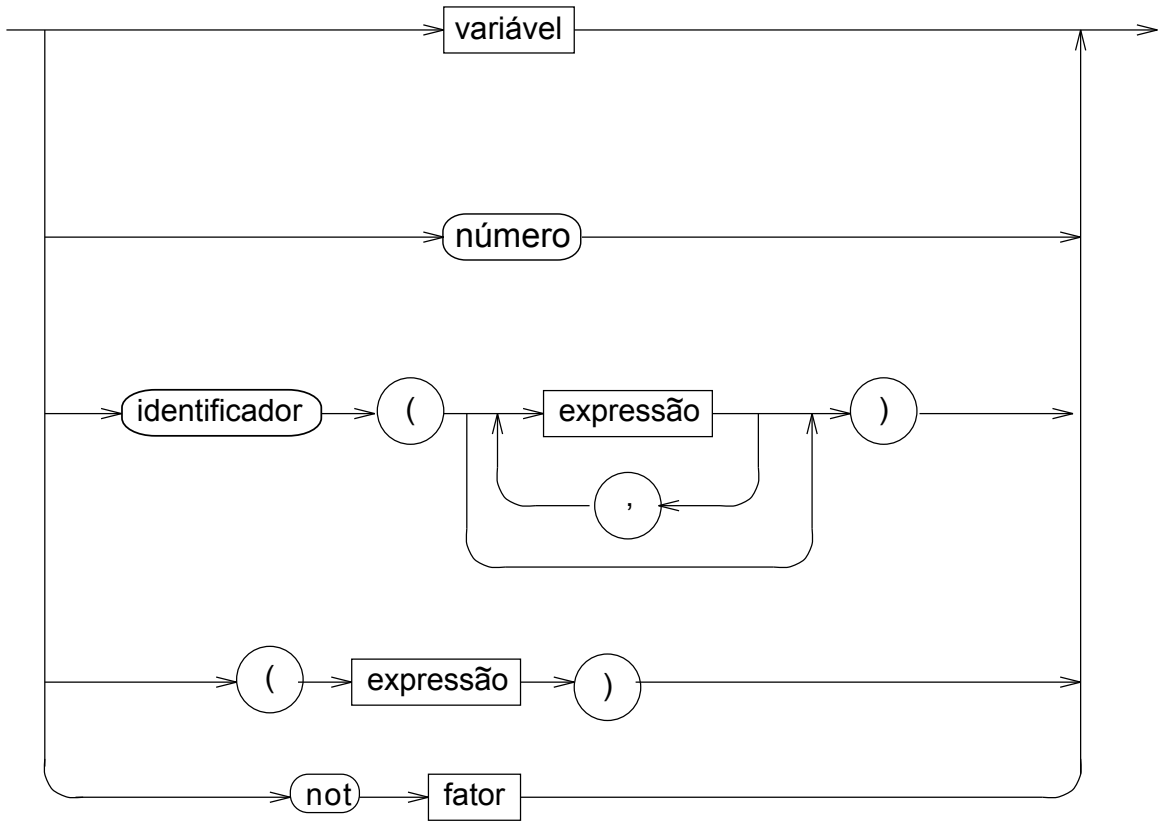
expressão simples:



termo:



fator:



constante: