**Poder Executivo**
**Ministério da Educação**
**Universidade Federal do Amazonas**
**Instituto de Computação**

UFAM

Disciplina: ICC043/IEC582 – Paradigmas de Linguagens de Programação 19/09/2019
Prof. Rafael Giusti (rgiusti@icomp.ufam.edu.br)

# Lista de Exercícios 3
# Gramáticas e Sintaxe de Linguagens de Programação

1) Qual é a diferença entre uma sentença e uma forma sentencial?

2) Qual é a diferença entre uma cadeia e uma sentença?

3) Assumindo que `[0-9]` é uma forma simplificada de escrever a expressão regular `(0+1+2+...+9)` e que `[a-z]` descreve a expressão `(a+b+...+z)`, quais são as linguagens geradas pelas expressões regulares a seguir?

   A. `(1(0+1)*1)+(0(0+1)*0)`

   B. `([0-9]*.[0-9][0-9]*)+([0-9][0-9]*.[0-9]*)`

   C. `[a-z](a+e+i)r`

   D. `([a-z]+[A-Z]+_)([a-z]+[A-Z]+[0-9]+_)*`

4) Quais linguagens abaixo não podem ser descritas por uma expressão regular? Quais não podem ser descritas por uma gramática livre de contexto?

   A. $L = \{a^n b^m \mid n > 0, m > 0\}$

   B. $L = \{a^n b^n \mid n > 0\}$

   C. $L = \{a^n b^n a^n \mid n > 0\}$

   D. $L = \{a^n a^n b^n \mid n > 0\}$

   E. O conjunto das literais de inteiros, strings e pontos flutuantes em Python.

   F. O conjunto de todos os programas sintaticamente válidos em uma linguagem hipotética que não possui tipos de dados e não exige que variáveis sejam delcaradas.

   G. O conjunto de todos os programas sintaticamente válidos em C ou em Python.

   H. O conjunto de todas as expressões algébricas bem formadas que podem ser escritas apenas com números decimais e os operadores de soma, subtração, multiplicação, divisão e parênteses.

   I. O conjunto de todas as equações $x$ = <expr> nas quais <expr> é uma expressão algébrica conforme descrito no item H.

   J. O conjunto de todas as equaçoes $x$ = <expr> nas quais, após a avaliação de <expr>, $x$ tem valor 0.

5) Faça uma gramática livre de contexto para sequências corretamente balanceadas de parênteses, colchetes e chaves. Uma sequência corretamente balanceada tem um casamento perfeito entre a abertura e o fechamento de um mesmo símbolo.

   Balanceado: "()", "()()", "(())", "(()())", "[()]", "([])", "{[()]}", "[({})]"
   Não balanceado: ")(", "(]", "[(])"

6) Reescreva a gramática do exercício anterior utilizando notação BNF ou EBNF.

7) Considere a gramática a seguir, em notação BNF.

```
<programa> ::= "begin" <comandos> "end"

<comandos> ::= <comando> | <comando> ";" <comandos>

<comando> ::= <atribuição> | <comando-if> | <comando-write>
           | "begin" <comandos> "end"

<atribuição> ::= <identificador> ":=" <expressão>

<comando-if> ::= "if" <expressão> "then" <comando>
               | "if" <expressão> "then" <comando> "else" <comando>

<comando-write> ::= "WriteLn" "(" <lista-write> ")"

<lista-write> ::= <expressão> | <expressão> "," <lista-write>
```

A. Complete a gramática, definindo os não-terminais restantes.

B. Encontre uma derivação mais à esquerda para a sentença abaixo:

```
begin a := 5; if a < 10 then WriteLn('Sim') else WriteLn('Não') end
```

C. Encontre uma derivação mais à direita para a mesma sentença.

D. Desenhe a árvore de derivação para a sentença.

E. Mostre que a gramática é ambígua.

F. Reescreva a regra `<comando-if>` para eliminar a ambiguidade.

8) Considere o seguinte programa em C.

```
1  #include <stdio.h>
2
3  int *funcao(void) {
4      int vlocal = 42;
5      return &vlocal;
6  }
7
8  int main(void) {
9      int *ponteiro = funcao();
10     printf("%d\n", *ponteiro);
11     return 0;
12 }
```

Existe um erro na linha 5. É possível escrever uma gramática livre de contexto que não permite esse tipo de construção na linguagem? Em outras palavras, é possível escrever uma gramática livre de contexto que permita escrever programas válidos na linguagem de programação C, mas não permita derivar cadeias que contenham o mesmo erro do programa acima? Jutifique.

9) Para os próximos itens, considere uma Simples Linguagem de Programação (SLiP). Na SLiP, todos os programas sequências de dois tipos de comandos:

- Comandos de declaração, na forma tipo `identificador = valor`, em que uma variável é declarada com um certo tipo e tem seu valor inicial definido;
- Comandos de atribuição, na forma `identificador = expressão`, em que o valor de uma variável passa a ser o valor da expressão.

As restrições da SLiP são:

- Os nomes das variáveis são uma única letra maiúscula (A, B, C, …, Z);
- Variáveis e expressões podem ser do tipo `int` ou do tipo `float`;
- Expressões podem ser um único literal (por exemplo, `42` ou `3.14`) ou uma soma de duas variáveis;
- Todos os comandos em SLiP são finalizados por ponto-e-vírgula.

Todos os comandos são finalizados por ponto-e-vírgula. O comportamento de um programa válido em SL é o seguinte:

- Todas as linhas de comandos são executadas sequencialmente;
- Ao final da execução os valores de todas as variáveis são mostrados.

Exemplo de um programa válido em SL:

| Programa | Saída |
|---|---|
| `int A = 10;`<br>`float B = 3.14;`<br>`B = A + B;` | A = 10<br>B = 13.14 |

Com base nessas especificações, faça

A) Escreva uma gramática léxica para SLiP, definindo os tokens <id>, <int> e <float>. Use expressões regulares, BNF, EBNF ou notação formal de gramáticas.

B) Escreva uma gramática sintática, em notação BNF ou EBNF, para SLiP.

Dica: comece com as três seguintes regras

1. `<programa> ::= <lista_comandos>`
2. `<lista_comandos> ::= <comando> “;”`
3. `<lista_comandos> ::= <comando> “;” <lista_comandos>`

C) Monte as árvores de derivação para as seguintes sentenças:

```
int A = 0 ;
int A = 3.14 ;

int A = 42 ; int B = 7 ; A = A + B ;
```