


1. Considere a seguinte gramática, tendo em conta que ID e LITERAL são tokens.

```
««  
g : ps  
ps : ps p ';' | p ';' ;  
p : fp ops  
fp : ID ':' rhs  
ops: ops '|' rhs  
    |  
rhs: rhs symb  
    |  
symb : ID  
      | LITERAL  
»»
```

Selecione a afirmação verdadeira:

2 POINTS

- 7/22 **A** Esta gramática tem 7 produções.
- 9/22 **B** Esta gramática tem 7 símbolos não-terminais.
- 4/22 **C** Esta gramática tem 2 símbolos terminais. 
- 2/22 **D** Esta gramática diz-se "Mista" pois as suas produções usam mais do que um tipo de recursividade.

2. Considere a seguinte gramática, sabendo que "g" é o axioma e tendo em conta que ID e LITERAL são tokens.

««

g : ps

ps : ps p ';' | p ';' |

p : fp ops

fp : ID ':' rhs

ops: ops '|' rhs

|

rhs: rhs symb

|

symb : ID

| LITERAL

»»

Selecione a afirmação que não é verdadeira:

2 POINTS

8/23 **A** A linguagem definida por esta gramática aceita a frase vazia.

2/23 **B** O símbolo "ps" representa uma lista em que o ";" não é um separador de elementos.

5/23 **C** Qualquer frase da linguagem definida por esta gramática corresponderá a uma Árvore de Derivação com raiz "g".

8/23 **D** Todos os elementos da lista "ps" dão origem a subfrases que contém exatamente um símbolo ":"

3. Considere a seguinte gramática G, tendo em conta o token ID representa uma sequência de letras (ex: ola), e o token LITERAL representa um único carácter entre " (ex: ':'). Assuma também que o analisador léxico ignora espaços, tabs e newlines.

```
««
g : ps
ps : ps p ';'
    | p ';'
p : fp ops
fp : ID ':' rhs
ops: ops '|' rhs
    |
rhs: rhs symb
    |
symb : ID
      | LITERAL
»»
```

Selecione a afirmação que é verdadeira:

2 POINTS

- 10/23 **A** A frase abaixo pertence à linguagem definida por G
- ```
E : E '|' T
 | T ;
T : T '&' F ;
T : F ;
F : int ;
```
- 3/23 **B** A frase abaixo pertence à linguagem definida por G
- ```
E : E '||' T
    | T ;
T : T '&&' F
    | F ;
F : int ;
```
- 6/23 **C** A frase abaixo pertence à linguagem definida por G
- ```
E : E '|' T ;
 | T ;
T : T '&' F ;
 | F ;
F : int ;
```
- 4/23 **D** - A frase abaixo não pertence à linguagem definida por G
- ```
E : E '|' T ;
E : T ;
T : T '&' F ;
T : F ;
F : int ;
```

4. Considere a seguinte gramática G, sabendo que "g" é o axioma e tendo em conta que ID e LITERAL são tokens.

Note que cada linha foi numerada no início (à esquerda) para mais fácil referencia nas questões.

««

01) g : ps

02) ps : ps p ';' p

03) | p ';' p

04) p : fp ops

05) fp : ID ':' rhs

06) ops: ops '|' rhs

07) |

08) rhs: rhs symb

09) |

10)symb : ID

11) | LITERAL

»»

Selecione a afirmação que não é verdadeira:

2 POINTS

- 9/23 **A** A linguagem gerada por G seria a mesma se as linhas 02) e 03) fossem trocadas por
02) ps : ps ';' p
03) | p
- 8/23 **B** A linguagem gerada por G seria a mesma se as linhas 08) e 09) fossem trocadas por
08) rhs : symb rhs
09) |
- 1/23 **C** A linguagem gerada por G seria a mesma se as linhas 10) e 11) fossem trocadas por
10)symb : LITERAL
11)symb : ID
- 5/23 **D** A linguagem gerada por G seria diferente se as linhas 06) e 07) fossem trocadas por
06) ops : rhs '|' ops
07) | rhs

5. Considere o conjunto de Símbolos Terminais $T=\{a, f, d, p, i\}$ e as seguintes frases válidas de uma linguagem L

a i f
a d p d i f
a i p i p i f
a d i p i f

Diga então se a afirmação abaixo é verdadeira:

««

L pode ser gerada pela seguinte GIC:

Frase \Rightarrow a Corpo f

Corpo \Rightarrow Ds Is

Ds \Rightarrow €

Ds \Rightarrow Ds p d

Is \Rightarrow i Resto

Resto \Rightarrow € | p i Resto

»»

2 POINTS

7/23 ☐ True

16/23 ☒ False

6. Observe com atenção o seguinte filtro de texto implementado com o Lex do Python

```
««
import ply.lex as lex
import sys
states = (('in','exclusive'),)
tokens = ( 'MA', 'MF', 'AQUI', 'IGN' )
def t_MA(t):
    r'\{[^}]+\}'
    t.lexer.begin('in')
def t_IGN(t):
    r'\n'
def t_in_MF(t):
    r'\{\{[^}]+\}'
    t.lexer.begin('INITIAL')
def t_in_AQUI(t):
    r'\n'
    print(t.value)
t_ANY_ignore = "
def t_ANY_error(t):
    t.lexer.skip(1)
lexer = lex.lex()
for linha in sys.stdin:
    lexer.input(linha)
    tok = lexer.token()
    while tok:
        tok = lexer.token()
»»
```

Selecione então a alínea abaixo que é uma afirmação verdadeira:

2 POINTS

- 9/22 **A** Se o texto de entrada for "agora {aqui}sim vai{{b} fica" o resultado seria 1 linha com "sim vai"
- 8/22 **B** Se o texto de entrada for "agora {aqui}sim vai{{b} fica" o resultado seria uma frase vazia (linha sem caracteres).
- 4/22 **C** Se o texto de entrada for "agora {aqui}sim vai{{b} fica" o resultado seriam 7 linhas com cada um dos caracteres "sim vai"
- 1/22 **D** Se o texto de entrada for "agora {aqui}sim vai{ab}adeus{{ab} fica" o resultado seria 1 linha com "adeus".

7. Observe abaixo, com cuidado, o Analisador Léxico (AL) para a linguagem B (LinB), que lhe deve ser já familiar, e que foi implementado com o Lex do PLY.

```
««
import ply.lex as lex
states = [ ('VALUE', 'exclusive') ]
literals = [',','{','}']
tokens = ( 'ID','PAL','TEXT','SEP' )
t_ID = r'(?i)@article|@book|@manual|@proceedings'
t_PAL = r'[a-zA-Z][a-zA-Z0-9]*'
def t_SEP(t):
    r '='
    t.lexer.begin('VALUE')
    return t
def t_VALUE_TEXT(t):
    r '\"[^\"]*\"|\\{.*\\}'
    t.lexer.begin('INITIAL')
    return t
t_ANY_ignore = ' \n\t'
def t_ANY_error(t):
    print('Illegal character: ', t.value[0])
    t.lexer.skip(1)
lexer = lex.lex()
»»
```

Escolha então a afirmação verdadeira nas alíneas abaixo:

2 POINTS

- 5/23 **A** Para este AL reconhecer corretamente todos os terminais da LinB, foi necessário definir dois estado internos de reconhecimento para além do default "INITIAL".
- 1/23 **B** Ao correr o AL acima, o compilador de Python assinala um erro porque considera o símbolo ANY (usado em "t_ANY_ignore" e em "t_ANY_error") como símbolo ou estado desconhecido.
- 7/23 **C** Ao ler um caracter igual '=', em qualquer ponto, o AL acima retorna sempre o símbolo "SEP".
- 10/23 **D** Ao ler no ficheiro de entrada o texto "BOOK", o AL acima não retorna o símbolo "ID".

8. Considere o seguinte Analisador Léxico (AL) especificado em PLY abaixo:

```
««
tokens = ["NUM"]
states = [ ("flip", "exclusive") ]
def t_NUM(t):
    r'\d+'
    t.lexer.count += 1
    return t
def t_newline(t):
    r'\n'
    t.lexer.begin("flip")
def t_flip_NUM(t):
    r'\d+\\.d+'
    t.lexer.count += 1
    return t
def t_flip_newline(t):
    r'\n'
    t.lexer.begin("INITIAL")
def t_ANY_other(t):
    r'.'
    pass
t_ANY_ignore = " \t"
def t_ANY_error(t):
    print("Illegal character")
    t.lexer.skip(1)

lexer = lex.lex()
lexer.count = 0
»»
```

Selecione então a opção correta:

2 POINTS

- 4/23 **A** Se ao analisador léxico gerado passarmos o input abaixo, no final o campo `lexer.count` terá o valor de 5.

```
««
O João tinha 5 maçãs, e deu 2 à Maria.
A Maria comeu 1.5 maçãs.
O que aconteceu às restantes 0.5 maçãs?
»»
```

- 5/23 **B** Se ao analisador léxico gerado passarmos o input abaixo, no final o campo `lexer.count` terá o valor de 4

```
««
O João tinha 5 maçãs, e deu 2 à Maria.
A Maria comeu 1.5 maçãs.
O que aconteceu às restantes 0.5 maçãs?
»»
```


7/23 C Se ao analisador léxico gerado passarmos o input abaixo, no final o campo ``lexer.count`` terá o valor de 4

««
O João tinha 5 maçãs, e deu 2 à Maria.
A Maria comeu 75% das suas maçãs.
O que aconteceu aos restantes 25%?
»»

7/23 D Se ao analisador léxico gerado passarmos o input abaixo, no final o campo ``lexer.count`` terá o valor de 3

««
O João tinha 5 maçãs, e deu 2.5 maçãs à Maria.
A Maria comeu 1.5 maçãs.
O que aconteceu à restante maçã?
»»

9. Observe abaixo, com cuidado, o Analisador Léxico (AL) para a linguagem B (LinB), que lhe deve ser já familiar, e que foi implementado com o Lex do PLY.

««

```
import ply.lex as lex
states = [ ('VALUE', 'exclusive') ]
literals = [ ',', '{', '}' ]
tokens = ( 'ID','PAL','TEXTO','SEP' )
t_ID = r'(?i)@article|@book|@manual|@proceedings'
t_PAL = r'[a-zA-Z][a-zA-Z0-9]*'
def t_SEP(t):
    r=' '
    t.lexer.begin('VALUE')
    return t
def t_VALUE_TEXTO(t):
    r'\("[^"]*"|\'[^\']*\'|\\{.*\\})'
    t.lexer.begin('INITIAL')
    return t
t_ANY_ignore = ' \n\t'
def t_ANY_error(t):
    print('Illegal character: ', t.value[0])
    t.lexer.skip(1)
lexer = lex.lex()
»»
```

Escolha então a afirmação que não está correta.

2 POINTS

- 6/23 A Se texto de entrada for
« **author = {{Vitor T. Martins} and Pedro Henriques}, »**
o AL acima retorna 4 símbolos terminais.
- 6/23 B Se texto de entrada for
« **'author' = "{Vitor T. Martins} and Pedro Henriques", »**
o AL acima retorna 4 símbolos terminais e 2 erros.
- 6/23 C Se texto de entrada for
« **@Article{Martins} »**
o AL acima retorna os símbolos terminais **"ID { PAL }"**.
- 5/23 D Se a ER `r'\("[^"]*"|\'[^\']*\'|\\{.*\\})'`
for trocada por `r'\("[^"]*"|\'[^\']*\'|\\{.*\\})'`
o processamento da frase de entrada
« **author = {{Vitor T. Martins} and Pedro Henriques}, »**
não se altera.

10. Considere o seguinte Analisador Léxico (AL) especificado em PLY abaixo:

```
««
tokens = ["NUM"]
states = [ ("flip", "exclusive") ]
def t_NUM(t):
    r'\d+'
    t.lexer.count += 1
    return t
def t_newline(t):
    r'\n'
    t.lexer.begin("flip")
def t_flip_NUM(t):
    r'\d+\\.d+'
    t.lexer.count += 1
    return t
def t_flip_newline(t):
    r'\n'
    t.lexer.begin("INITIAL")
def t_ANY_other(t):
    r'.'
    pass
t_ANY_ignore = " \t"
def t_ANY_error(t):
    print("Illegal character")
    t.lexer.skip(1)
```

```
lexer = lex.lex()
lexer.count = 0
```

»»

e diga se a afirmação seguinte é verdadeira ou falsa:

««

Caso se trocasse na regra de produção

```
def t_NUM(t):
```

a linha

```
    t.lexer.count += 1
```

por

```
    t.lexer.count += int(t.value)
```

o comportamento do AL ao processar as linhas pares do input era alterado.

»»

2 POINTS

17/22 ☒ True

5/22 ☐ False