Lista de Linguagens de Programação – 3

Nome: _____ Matrícula: _____

1. Insira parênteses no comando abaixo, de acordo com a precedência dos operadores:

   ```
   a = b < c ? * p + b * c : 1 << d ()
   ```

2. Esta questão refere-se à gramática abaixo, que representa uma linguagem muito simples, de somas de números. Por simplicidade nós não mostraremos as regras de produção para números:

$$\begin{array}{rcl} \langle E \rangle & ::= & \langle E \rangle + \langle E \rangle \\ & | & \langle Number \rangle \end{array}$$

   (a) Prove que a gramática em questão é ambígua.

   (b) Mostre como esta ambiguidade compromete a semântica da linguagem que a gramática representa.

   (c) Forneça uma gramática que reconheça a mesma linguagem, mas que não seja ambígua.

3. As questões a seguir referem-se ao conceito de *associatividade* de operadores.

   (a) O que significa dizer que um operador é associativo à esquerda ou à direita?

   (b) Considere o operador de soma aritmética +, usado na linguagem C. Este operador é associativo à esquerda ou à direita?

   (c) Dê um exemplo de um operador em C que seja associativo à direita.

   (d) Modifique a gramática abaixo para que operadores sejam associativos á esquerda:

```
<exp>     ::= <mulexp> + <exp>
            |  <mulexp>
<mulexp>  ::= <rootexp> * <mulexp>
            |  <rootexp>
<rootexp> ::= ( <exp> )
            |  <number>
```

4. Considere a gramática abaixo, usada no exercício anterior:

```
<exp>     ::= <mulexp> + <exp>
           |  <mulexp>
<mulexp>  ::= <rootexp> * <mulexp>
           |  <rootexp>
<rootexp> ::= ( <exp> )
           |  <number>
```

   (a) Escreva esta gramática em Prolog.

   (b) Modifique a gramática construída no exercício anterior para que ela compute o valor da expressão aritimética. Por exemplo, o predicado `expr(N, [2,*,'(',3,+,4,')'], []).`, é verdade se `N = 14`.

5. Este exercício solidifica alguns conceitos relacionados à hierarquia de gramáticas proposta por Noam Chomsky.

   (a) As gramáticas mais simples são chamadas *Gramáticas Regulares*. Estas gramáticas possuem três regras de produação muito rudimentares: um não-terminal produz um terminal seguido de um não terminal, ou um terminal, ou a palavra vazia. Embora muito simples, estas gramáticas são muito úteis. Por exemplo, elas são usadas pelos compiladores para separar um arquivo de entrada em palavras independentes, isto é, números, palavras-chave, identificadores, etc. Converta a gramática abaixo para a forma regular:

```
number --> digit.
number --> digit, number.
digit --> [0] ; [1] ; [2] ; [3] ; [4] ; [5] ; [6] ; [7] ; [8] ; [9].
```

   (b) A linguagem $\{a^n b^n \mid n \in N\}$ não pode ser reconhecida por uma gramática regular, mas ela pode ser reconhecida por uma gramática *livre de contexto*. Construa uma gramática que reconheça esta linguagem em Prolog.

   (c) Há muitas linguagens que não são livres de contexto. Por exemplo, não é possível usar uma gramática destas para verificar se um programa usa as variáveis com os tipos corretos. Tampouco é possível reconhecer a linguagem $\{a^n b^n c^n \mid n \in N\}$ via uma gramática livre de contexto. Ainda assim é possível reconhecer esta linguagem em Prolog, via atributos. Escreva uma gramática, em Prolog, que reconheça esta linguagem.