Aula 4: Lista

Definição 1: No Prolog, a lista é heterogênea, ou seja, permite que qualqer tipo de dado seja armazenado em sua estrutura.

Exemplo: [mia, 2, ponto(4, -2), X, [2,1]]

Definição 2: No Prolog, assim como nas outras linguagens, o menor tamanho de uma lista é zero, implicando em uma lista vazia.

Exemplo: []

Definição 3: No Prolog, toda lista não vazia pode ser vista como uma lista contendo duas partes: cabeça e cauda. A cabeça da lista é representada pelo primeiro elemento da lista. A cauda da lista é representada por todo o restante, menos a cabeça.

Exemplos:

Considerando a lista **[mia, vicent, yolanda, bob]**, quem é a cabeça e quem é a calda?

Considerando a lista **[mia]**, quem é a cabeça e quem é a calda?

Definição 3: Para extrair elementos de uma lista, utilizamos o operador |

Exemplo:

Seja a lista **[mia, vicent, yolanda, bob]**, extraímos a cabeça e

calda da seguinte forma:

In [ ]:
```
[Head | Tail] = [mia, vicent, yolanda, bob]
```

Nesse exemplo, a variável **Head** vai armazenar o termo **mia** e a variável **Tail** a sublista **[vicent, yolanda, bob]**.

Exemplo

Seja a lista **[mia, vicent, yolanda, bob]**, podemos extrair mais de um elemento na cabeça da seguinte forma:

In [ ]:
```
[X, Y | Z] = [mia, vicent, yolanda, bob]
```

Nesse exemplo, quais os valores das variáveis X, Y e Z?

Exemplo

Seja a lista **[mia, vicent, yolanda, bob]**, podemos extrair um elemento em uma posição específica da lista da seguinte forma:

In [ ]:
```
[_, X | Z] = [mia, vicent, yolanda, bob]
```

Nesse exemplo, estamos utilizando uma variável anônima para ignorar o elemento da primeira posição da lista e definindo a variável X para armazenar o elemento da segunda posição da lista. A calda está sendo armazenada em Z.

Agora que já sabemos como o operador | funciona, vamos construir alguns programas de manipulação básica:

P1: Somar os elementos de uma lista.

Versão 1

In [ ]:
```
soma(L, R) :- soma(L, 0, R).
soma([], R, R) :- !.
soma([H | T], Acc, R) :- A is H+Acc,
                         soma(T, A, R).
```

Versão 2

In [ ]:
```
soma([], 0).
soma([H | T], N) :- soma(T, N1), N is N1 + H.
```

P2: Calcular o tamanho de uma lista.

Versão 1

In [ ]:
```
tamanho(L, R) :- tamanho(L, 0, R).
tamanho([], R, R) :- !.
tamanho([_ | T], Acc, R) :- A is 1+Acc,
                            tamanho(T, A, R).
```

Versão 2

In [ ]:
```
tamanho([], 0).
tamanho([_ | T], N) :- tamanho(T, N1), N is N1 + 1.
```

P3: Verificar se um elemento está na lista.

In [ ]:
```
contem(H, [H | _]) :- !.
contem(H, [_ | T]) :- contem(H, T).
```

P4: Encontrar o maior valor da lista.

Versão 1

In [ ]:
```
maior([X | T], R) :- maior(T, X, R).
maior([], R, R) :- !.
maior([X | T], Max, R) :- Max1 is max(X, Max),
                          maior(T, Max1, R).
```

Versão 2

In [ ]:
```
maior([X], X) :- !.
maior([X | T], Max) :- maior(T, Max1), Max is max(X, Max1).
```

P5: Recuperar um elemento pelo índice.

In [ ]:
```
recuperar_por_indice([X | _], 0, X) :- !.
recuperar_por_indice([_ | Xs], I, E) :- Ix is I-1,
                                        recuperar_por_indice(Xs, Ix,
```

P6: Recuperar o índice de um elemento.

Versão 1

In [ ]:
```
recuperar_indice(L, E, R) :- recuperar_indice(L, E, 0, R).
recuperar_indice([E | _], E, R, R) :- !.
recuperar_indice([_ | Xs], E, Acc, R) :- Acc1 is Acc + 1,
                                         recuperar_indice(Xs, E, Acc1
```

Versão 2

In [ ]:
```
recuperar_indice([E | _], E, 0) :- !.
recuperar_indice([_ | Xs], E, I) :- recuperar_indice(Xs, E, I1),
                                    I is I1 + 1.
```

P7: Somar duas listas dois a dois.

Versão 1

In [ ]:
```
somar([], [], []).
somar(L1, L2, R) :- somar(L1, L2, [], R).

somar([], [], R, R) :- !.
somar([X | L1], [Y | L2], AUX, R) :- Z is X + Y,
                                     somar(L1, L2, [Z | AUX], R).
```

Essa **versão 1** tem um problema, você consegue identificar qual e por quê?

Versão 2

In [ ]:
```
somar([], [], []).
somar(L1, L2, R) :- somar(L1, L2, [], R).

somar([], [], R, R) :- !.
somar([X | L1], [Y | L2], AUX, R) :- Z is X + Y,
                                     append(AUX, [Z], AUX1),
                                     somar(L1, L2, AUX1, R).
```

Na **versão 2**, estamos corrigindo o problema da **versão 1**, utilizando a regra **append**.

Versão 3

In [ ]:
```
somar([], [], []).
somar([X | Xs], [Y | Ys], [Z | Zs]) :- Z is X + Y,
                                        somar(Xs, Ys, Zs).
```

Exercício

Q1: Regra para verificar a frequência de um elemento na lista.

Q2: Regra para concatenar duas listas.

Q3: Regra para concatenar duas listas dois a dois.

Q4: Regra para somar as listas que estão dentro de uma lista. Exemplo: [[1,2,3],[1,1,1]] gera a saída [6, 3].

Q5: Regra para encontrar o último elemento na lista.

Q6: Regra para contar elementos contíguos 2 a 2 em uma lista.

Q7: Regra para construir a lista reversa.

Q8: Regra para informar se o elemento está na lista.

Q9: Regra para informar os elementos duplicados, considerando a ordem em que eles aparecem na lista.

Q10: Regra para eliminar os elementos duplicados.