Aula Prática 2

Recursividade e Listas

Objetivos:
- Recursividade em Prolog
- Listas em Prolog

1. Recursividade

a) Implemente o predicado *fatorial(+N, ?F)*, que calcula o fatorial de um número *N*.

b) Implemente o predicado *somaRec(+N, ?Sum)*, uma versão recursiva do cálculo do somatório dos números de 1 até *N*.

c) Implemente o predicado *fibonacci(+N, ?F)*, que calcula o número de fibonacci de ordem *N*.

d) Implemente o predicado *isPrime(-X)*, que determina se *X* é um número primo. Sugestão: um número é primo se for divisível apenas por si próprio e por 1.

Fatorial, somatório, série de Fibonacci e primalidade de alguns valores:

N	0	1	2	3	4	5	6
Fatorial	0	1	2	6	24	120	720
Somatório	0	1	3	6	10	15	21
Fibonacci	0	1	1	2	3	5	8
Primo	no	yes	yes	yes	no	yes	no

2. Relações Familiares

Considere a base de factos do exercício 1 da ficha da aula anterior sobre relações familiares.

a) Implemente o predicado *ancestor(?X, ?Y)* que suceda se X é um antepassado de Y.

b) Implemente o predicado *descendant (?X, ?Y)* que sucede se X é descendente de Y.

3. Cargos e Chefes

Considere a base de factos do exercício 5 da ficha da aula anterior sobre cargos e chefes. Implemente o predicado *superior(+X, +Y)* que suceda se a pessoa X ocupa um cargo superior ao cargo ocupado pela pessoa Y.

4. Funcionamento de Listas

Sem usar o interpretador, indique o resultado de cada uma das seguintes igualdades em Prolog:

```
a) | ?- [a | [b, c, d] ] = [a, b, c, d]
b) | ?- [a | b, c, d ] = [a, b, c, d]
c) | ?- [a | [b | [c, d] ] ] = [a, b, c, d]
```

```
d) | ?- [H|T] = [pfl, lbaw, redes, ltw]
e) | ?- [H|T] = [lbaw, ltw]
f) | ?- [H|T] = [leic]
g) | ?- [H|T] = []
h) | ?- [H|T] = [leic, [pfl, ltw, lbaw, redes] ]
i) | ?- [H|T] = [leic, Two]
j) | ?- [Inst, feup] = [gram, LEIC]
k) | ?- [One, Two | Tail] = [1, 2, 3, 4]
l) | ?- [One, Two | Tail] = [leic | Rest]
```

5. Recursividade sobre listas

a) Implemente o predicado *list_size(+List, ?Size)* que determina o tamanho de *List*.

b) Implemente o predicado *list_sum(+List, ?Sum)* que soma os valores contidos em *List*.

c) Implemente o predicado *list_prod(+List, ?Prod)* que multiplica os valores contidos em *List*.

d) Implemente o predicado *inner_product (+List1, +List2, ?Result)* que calcula o produto interno de dois vetores (representados como listas de inteiros, do mesmo tamanho).

e) Implemente o predicado *count(+Elem, +List, ?N)*, que conta o número de ocorrências de *Elem* em *List*, colocando o resultado em *N*.

6. Manipulação de Listas

a) Implemente o predicado *invert(+List1, ?List2)*, que inverte a lista *List1*.

b) Implemente o predicado *del_one(+Elem, +List1, ?List2)*, que apaga uma ocorrência de *Elem* de *List1*, resultando em *List2*.

c) Implemente o predicado *del_all(+Elem, +List1, ?List2)*, que apaga todas as ocorrências de *Elem* de *List1*, resultando em *List2*.

d) Implemente o predicado *del_all_list(+ListElems, +List1, ?List2)*, que apaga de *List1* todas as ocorrências de todos os elementos de *ListElems*, resultando em *List2*.

e) Implemente o predicado *del_dups(+List1, ?List2)*, que elimina valores repetidos de *List1*.

f) Implemente o predicado list_*perm (+L1, +L2)* que sucede se *L2* for uma permutação de *L1*.

g) Implemente o predicado *replicate(+Amount, +Elem, ?List)* que gera uma lista com *Amount* repetições de *Elem*.

h) Implemente o predicado *intersperse(+Elem, +List1, ?List2)* que intercala *Elem* entre os elementos de *List1*, devolvendo o resultado em *List2*.

i) Implemente o predicado *insert_elem(+Index, +List1, +Elem, ?List2)*, que insere *Elem* em *List1* no índice *Index*, daí resultando *List2*.

j) Implemente o predicado *delete_elem(+Index, +List1, ?Elem, ?List2)*, que remove o elemento no índice *Index* de *List1* (que é unificado com *Elem*), resultando em *List2*.

   Como compara a implementação deste predicado com o anterior? Seria possível usar um único predicado para realizar as duas operações? De que forma?

k) Implemente o predicado *replace(+List1, +Index, ?Old, +New, ?List2)*, que substitui o elemento *Old*, localizado no índice *Index* de *List1*, por *New*, resultando em *List2*.

7. Append, O Poderoso

a) Implemente o predicado *list_append(?L1, ?L2, ?L3)* em que *L3* é constituída pela concatenação das listas *L1* e *L2*.

b) Implemente o predicado *list_member(?Elem, ?List)* que verifica se *Elem* é membro de *Lista* usando unicamente o predicado *append* uma só vez.

c) Implemente o predicado *list_last(+List, ?Last)* que unifica *Last* com o último elemento de *List*, usando unicamente o predicado *append* uma só vez.

d) Implemente o predicado *list_nth(?N, ?List, ?Elem)*, que unifica *Elem* com o *N*-ésimo elemento de *Lista*, usando apenas os predicados *append* e *length*.

e) Implemente o predicado *list_append(+ListOfLists, ?List)* que concatena uma lista de listas.

f) Implemente o predicado *list_del(+List, +Elem, ?Res),* que elimina uma ocorrência de *Elem* de *List*, unificando o resultado com *Res*, usando apenas o predicado *append* duas vezes.

g) Implemente o predicado *list_before(?First, ?Second, ?List)* que sucede se os dois primeiros argumentos forem membros de *List*, e *First* ocorrer antes de *Second*, usando unicamente o predicado *append* duas vezes.

h) Implemente o predicado *list_replace_one(+X, +Y, +List1, ?List2)* que substitui uma ocorrência de *X* em *List1* por *Y*, daí resultando *List2*, usando unicamente o predicado *append* duas vezes.

i) Implemente o predicado *list_repeated(+X, +List)* que sucede se *X* ocorrer repetidamente (pelo menos duas vezes) em *List*, usando unicamente o predicado *append* duas vezes.

j) Implemente o predicado *list_slice(+List1, +Index, +Size, ?List2)*, que extrai uma fatia de tamanho *Size* de *List1* começando no índice *Index*, resultando em *List2*, usando apenas os predicados *append* e *length*.

k) Implemente o predicado *list_shift_rotate(+List1, +N, ?List2)*, que rode *List1 N* elementos para a esquerda, resultando em *List2*, usando apenas os predicados *append* e *length*.

Ex:
```
| ?- list_shift_rotate([a, b, c, d, e, f], 2, L).
   L = [c, d, e, f, a, b]
```

8. Listas de Números

a) Implemente o predicado *list_to(+N, ?List)*, que unifica *List* com uma lista com todos os números inteiros entre 1 e *N*.

b) Implemente o predicado *list_from_to(+Inf, +Sup, ?List)*, que unifica *List* com uma lista com todos os inteiros entre *Inf* e *Sup* (ambos incluídos).

c) Implemente o predicado *list_from_to_step(+Inf, +Step, +Sup, ?List)*, que unifica *List* com uma lista contendo os inteiros entre *Inf* e *Sup*, em incrementos de *Step*.

d) Altere as soluções das duas alíneas anteriores para detetar os casos em que *Inf* é superior a *Sup*, devendo nesses casos devolver a lista com os elementos em ordem decrescente (sugestão: considere usar inversão de listas).

e) Implemente o predicado *primes(+N, ?List)*, que unifica *List* com uma lista contendo todos os números primos até *N* (sugestão: use o predicado *isPrime*, do exercício 1).

f) Implemente o predicado *fibs(+N, ?List)*, que unifica *List* com uma lista com todos os números de Fibonacci de ordem 0 até *N* (sugestão: use o predicado do exercício 1).

9. Run-Length Encoding

a) Implemente o predicado *rle(+List1, ?List2)*, que faça a compressão *run-length* de *List1* colocando o resultado em List2 usando pares de valores.

b) Implemente o predicado *un_rle(+List1, ?List2)*, que faça a descompressão de *List1*.

Ex:
```
| ?- rle([a, a, b, b, b, c, c, c, d, d, e, f, f, f, g, g, g, g, g], L).
L = [a-2, b-3, c-3, d-2, e-1, f-3, g-5]
| ?- un_rle([a-2, b-3, c-3, d-2, e-1, f-3, g-7], L).
L = [a, a, b, b, b, c, c, c, d, d, e, f, f, f, g, g, g, g, g, g, g]
```

10. Ordenação de Listas

a) Implemente o predicado *is_ordered(+List)* que sucede se *List* é uma lista de inteiros ordenados de forma ascendente.

b) Implemente o predicado *insert_ordered(+Value, +List1, ?List2)*, que insere *Value* em *List1*, mantendo a ordenação dos elementos, colocando o resultado em *List2*.

c) Implemente o predicado *insert_sort(+List, ?OrderedList)*, que ordena *List*.

11. Triângulo de Pascal

Implemente o predicado *pascal(+N, ?Lines)*, em que *Lines* é uma lista com as primeiras *N* linhas do triângulo de Pascal.

```
      1
    1   1
  1   2   1
1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
```