Aula Prática 3

Cut, I/O e Predicados de Obtenção de Soluções Múltiplas

Objetivos:
- Perceber funcionamento e usar cut
- Entrada e Saída de Dados
- Predicados de Obtenção de Soluções Múltiplas

1. Funcionamento do Cut

Considere o seguinte código:

```
s(1).
s(2):- !.
s(3).
```

Sem usar o interpretador, indique qual o resultado de cada uma das seguintes interrogações:

```
a) | ?- s(X).

b) | ?- s(X), s(Y).

c) | ?- s(X), !, s(Y).
```

2. Efeito do Cut

Considere o seguinte código.

```
data(one).
data(two).
data(three).

cut_test_a(X):- data(X).
cut_test_a('five').

cut_test_b(X):- data(X), !.
cut_test_b('five').

cut_test_c(X, Y):- data(X), !, data(Y).
cut_test_c('five', 'five').
```

Sem usar o interpretador, indique o resultado de cada uma das seguintes interrogações.

```
a) | ?- cut_test_a(X), write(X), nl, fail.

b) | ?- cut_test_b(X), write(X), nl, fail.

c) | ?- cut_test_c(X, Y), write(X-Y), nl, fail.
```

3. Cuts Vermelhos e Verdes

Indique, justificando, se cada um dos cuts presentes no seguinte código é verde ou vermelho.

```
immature(X):- adult(X), !, fail.
immature(X).
adult(X):- person(X), !, age(X, N), N >=18.
adult(X):- turtle(X), !, age(X, N), N >=50.
adult(X):- spider(X), !, age(X, N), N>=1.
adult(X):- bat(X), !, age(X, N), N >=5.
```

4. Entrada e Saída de Dados

Implemente as seguintes alíneas sem usar o predicado *format/2*.

a) Implemente o predicado *print_n(+S, +N)* que imprime *N* vezes na consola o símbolo *S*.

b) Implemente o predicado *print_text(+Text, +Symbol, +Padding)* que imprime o texto presente no primeiro argumento (usando aspas) com o *padding* indicado no terceiro argumento (número de espaços antes e depois do texto), rodeado de *Symbol*. Ex.:

```
| ?- print_text("Olá Mundo!", '*', 4).
*    Olá Mundo    *
```

c) Implemente o predicado *print_banner(+Text, +Symbol, +Padding)* que imprime o texto presente no primeiro argumento (usando aspas) com o formato do exemplo abaixo:

```
| ?- print_banner("Olá Mundo!", '*', 4).
**********************
*                    *
*    Olá Mundo       *
*                    *
**********************
```

d) Implemente o predicado *read_number(-X)* que lê um número da entrada padrão, dígito a dígito (ie, sem usar *read*), devolvendo esse número (como inteiro). Sugestão: use *peek_code* para determinar quando terminar a leitura (o código ASCCI de *Line Feed* é 10).

e) Implemente o predicado *read_until_between(+Min, +Max, -Value)*, que peça ao utilizador para inserir um inteiro entre *Min* e *Max*, e retorne apenas quando o valor inserido estiver entre esses limites. Dica: garanta que o predicado *read_number/1* é determinista.

f) Implemente o predicado *read_string(-X)* que lê uma cadeia de caracteres da entrada padrão, caráter a caráter, devolvendo uma string (ie, uma lista de códigos ASCII).

g) Implemente o predicado *banner/0* que pede ao utilizador os argumentos a usar numa chamada ao predicado *print_banner/3*, lê esses argumentos, e invoca o predicado.

h) Implemente o predicado *print_multi_banner(+ListOfTexts, +Symbol, +Padding)* que imprime várias linhas de texto no formato de um *banner*, usando a linha de maior comprimento para cálculo do *padding* a usar nas restantes.

```
| ?- print_multi_banner(["Frase um", "Frase Dois"], '*', 4).
**********************
*                    *
*     Frase um       *
*    Frase Dois      *
*                    *
**********************
yes
| ?- print_multi_banner(["Olá Mundo! ", [73,32,9829,32,80,114,111,
108,111,103], [73,116,32,82,117,108,122,33]], '*', 4).
...
```

i) Implemente o predicado *oh_christmas_tree(+N)* que imprime uma árvore de tamanho *N*.

```
| ?- oh_christmas_tree(5).
     *
    ***
   *****
  *******
 *********
     *
```

5. Relações Familiares Revisitadas

Considere o exercício 1 da primeira ficha de exercícios, sobre relações familiares.

a) Implemente o predicado *children(+Person, -Children)*, que devolve no segundo argumento uma lista com todos os filhos de *Person*.

b) Implemente o predicado *children_of(+ListOfPeople, -ListOfPairs)*, que devolve no segundo argumento uma lista com pares no formato *P-C*, em que *P* é elemento de *ListOfPeople*, e *C* é uma lista com os seus filhos.

c) Implemente o predicado *family(-F)* que devolve uma lista com todas as pessoas da família.

d) Implemente o predicado *couple(?C)*, que unifica *C* com um par de pessoas (*X-Y*) que têm pelo menos um filho (descendente) em comum. Ex.:

```
| ?- couple(claire-phil).
yes
| ?- couple(C).
C = dede-jay ?
```

e) Implemente o predicado *couples(-List)* que devolve em *List* uma lista com todos os casais que tiveram filhos, evitando resultados duplicados.

f) Implemente o predicado *spouse_children(+Person -SC)* que devolve em *SC* um par *Spouse/Children* com um cônjuge e filhos de *Person* e *Spouse*.

g) Implemente o predicado *immediate_family(+Person, -PC)* que devolve em *PC* um par *A-B* em que *A* é uma lista com os progenitores de *Person*, e *B* é uma lista com os cônjuges e respetivos filhos de *Person*.

```
| ?- immediate_family(haley, X).
X = [phil,claire]-[dylan/[george,poppy]] ?
```

h) Implemente o predicado *parents_of_two(-Parents)* que devolve em *Parents* a lista de todas as pessoas que tiveram pelo menos dois filhos.

6. Professores e Alunos Revisitados

Considere o exercício 2 da primeira ficha de exercícios, sobre professores e alunos.

a) Implemente o predicado *teachers(-T)* que devolve uma lista com todos os professores.

b) Como se comportaria o predicado que implementou na alínea anterior caso um professor lecione mais de uma UC? Como poderia evitar duplicados?

c) Implemente o predicado *students_of(+T, -S)* que devolve uma lista com todos os estudantes do professor *T*.

d) Implemente o predicado *teachers_of(+S, -T)* que devolve uma lista com todos os professores do estudante *S*.

e) Implemente o predicado *common_courses(+S1, +S2, -C)*, que devolve uma lista de todas as unidades curriculares frequentadas por ambos os estudantes *S1* e *S2*.

f) Implemente o predicado *more_than_one_course(-L)* que devolve uma lista com todos os estudantes que frequentam mais de 1 UC. Nota: evite elementos duplicados.

g) Implemente o predicado *strangers(-L)* que devolve uma lista de todos os pares de estudantes que não se conhecem, ie, que não frequentam nenhuma UC em comum.

h) Implemente o predicado *good_groups(-L)* que devolve uma lista com todos os pares de estudantes que frequentam mais de uma UC em comum.

7. Horários

Considere o seguinte excerto de código representativo dos horários de uma turma da L.EIC:

```
%class(Course, ClassType, DayOfWeek, Time, Duration)
class(pfl, t, '1 Seg', 11, 1).          class(ltw, tp, '5 Sex', 8.5, 2).
class(pfl, t, '4 Qui', 10, 1).          class(fsi, t, '1 Seg', 12, 1).
class(pfl, tp, '2 Ter', 10.5, 2).       class(fsi, t, '4 Qui', 12, 1).
class(lbaw, t, '1 Seg', 8, 2).          class(fsi, tp, '3 Qua', 8.5, 2).
class(lbaw, tp, '3 Qua', 10.5, 2).      class(rc, t, '4 Qui', 8, 2).
class(ltw, t, '1 Seg', 10, 1).          class(rc, tp, '5 Sex', 10.5, 2).
class(ltw, t, '4 Qui', 11, 1).
```

a) Implemente o predicado *same_day(+UC1, +UC2)* que sucede se existem aulas de *UC1* e *UC2* que decorrem no mesmo dia.

b) Implemente o predicado daily_courses(+Day, -Courses) que recebe um dia da semana, e devolve uma lista com todas as UCs que têm aulas nesse dia.

c) Implemente o predicado *short_classes(-L)* que devolve em L uma lista de todas as aulas com duração inferior a 2h (lista de termos no formato *UC-Dia/Hora*).

d) Implemente o predicado *course_classes(+Course, -Classes)* que recebe uma UC e devolve uma lista com todas as aulas dessa UC (lista de termos no formato *Dia/Hora-Tipo*).

e) Implemente o predicado *courses(-L)* que devolve uma lista com todas as UCs existentes. Evite resultados repetidos no resultado.

f) Implemente o predicado *schedule/0* que imprime na consola todas as aulas, por ordem de ocorrência na semana.

g) Modifique os predicados anteriores de forma a que o dia da semana seja impresso apenas como *seg*, *ter*, *qua*, *qui* ou *sex*. Sugestão: use um predicado de 'tradução' para converter entre o formato de representação interna e o formato de visualização.

h) Implemente o predicado *find_class/0* que pede ao utilizador um dia e hora, e indica se alguma aula inicia ou decorre a essa hora, imprimindo a aula, hora de início e duração.