Practical Class 4

Cut and I/O

Objectives:
- Understand how the cut works, and use it
- Data Input and Output

1. How the Cut works

Consider the following code:

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

Without using the interpreter, state the result of each of the following queries:

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

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

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

2. The effect of a Cut

Consider the following code:

```
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').
```

Without using the interpreter, state the result of each of the following queries:

```
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. Red and Green Cuts

State, justifying, whether each of the cuts in the code is red or green.

```
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. Maximum Value

Implement *max(+A, +B, +C, ?Max)*, which determines the maximum value between three numbers. Note: avoid behaviors such as shown below:

```
| ?- max(2,3,3,X).
X = 3 ? ;
X = 3 ? ;
no
```

5. Data Input and Output

Implement the following predicates without using the *format/2* predicate.

a) Implement *print_n(+N, +S)* which prints symbol *S* to the terminal *N* times.

b) Implement *print_text(+Text, +Symbol, +Padding)* which prints the text received in the first argument (using double quotes) with the padding received in the third argument (number of spaces before and after the text), and surrounded by *Symbol*. Example:

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

c) Implement *print_banner(+Text, +Symbol, +Padding)* which prints the text received in the first argument (using double quotes) using the format of the example below:

```
| ?- print_banner("Hello World!", '*', 4).
**********************
*                    *
*    Hello World!    *
*                    *
**********************
```

d) Implement *read_number(-X)*, which reads a number from the standard input, digit by digit (i.e., without using *read*), returning that number (as an integer). Suggestion: use *peek_code* to determine when to terminate the reading cycle (the ASCCI code for *Line Feed* is 10).

e) Implement *read_until_between(+Min, +Max, -Value)*, which asks the user to insert an integer number between *Min* and *Max*, and succeeds only when the value inserted is within those limits. Hint: ensure that the *read_number/1* predicate is determinate.

f) Implement *read_string(-X)*, which reads a string of characters from the standard input, character by character, returning a string (i.e., a list of ASCII codes).

g) Implement *banner/0*, which asks the user for the arguments to use in a call to *print_banner/3*, reads those arguments, and invokes the predicate.

h) Implement *print_multi_banner(+ListOfTexts, +Symbol, +Padding)* which prints several lines of texto in the format of a *banner*, using the longest line to determine the padding to use in the remaining lines.

```
| ?- print_multi_banner(["Hello World", "Bye"], '*', 4).
**********************
*                    *
*    Hello World     *
*        Bye         *
*                    *
**********************
yes
| ?- print_multi_banner(["Hello World!", [73,32,9829,32,80,114,111,
108,111,103], [73,116,32,82,117,108,122,33]], '*', 4).
```

i) Implement *oh_christmas_tree(+N)* which prints a tree of size *N*.

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

6. List Printing

a) Implement *print_full_list(+L)*, which receives a list and prints it to the terminal using spaces in addition to commas to separate elements in the list.

b) Implement *print_list(+L)*, which receives a list and either prints it in full if its size is up to 11 elements, or prints the first, middle and last three elements of the list, plus ellipses in between these three groups for lists with 12 or more elements. Use the predicate from the previous question to print the lists with spaces between the elements. Examples:

```
| ?- print_list([a,b,c,d,e,f,g,h,i,j,k]).
[a, b, c, d, e, f, g, h, i, j, k]
yes
| ?- print_list([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p]).
[a, b, c, ..., h, i, j, ..., n, o, p]
yes
```

c) Implement print_*matrix(+M)*, which prints a list of lists, with each list in a line, and formatted as above. Example:

```
| ?- print_matrix([[a,b,c], [d,e,f], [g,h,i]]).
[a, b, c]
[d, e, f]
[g, h, i]
yes
```

d) Create a new version of the predicate above, *print_numbered_matrix(+L)*, that prints the matrix with a line number before each line. Note that the matrix may have more than nine lines, so the numbers should be padded to ensure a correct number alignment. Example:

```
| ?- length(_L, 100), maplist(=([a,b,c,d]), _L),
 print_numbered_matrix(_L).
   1 [a, b, c, d]
       ( ... )
  99 [a, b, c, d]
 100 [a, b, c, d]
yes
```

e) Implement *print_list(+L, +S, +Sep, +E)*, which prints a list using *S* as a starting symbol, *Sep* as the separator between elements, and *E* as the end symbol. Example:

```
| ?- print_list([a,b,c,d,e], '[', ', ', ']').
[a, b, c, d, e]
yes
| ?- print_list([a,b,c,d,e], '< ', ' | ', ' >').
< a | b | c | d | e >
yes
```

7. The Cut, Yet Again

How do you classify each of the cuts used in the solutions to the exercises above (as red/green)?