CS_205 Declarative Programming 3 December 2019
Monika Seisenberger

Assignment 2: Due 17 December 2019, 11am

1. This coursework will be submitted in small groups of three. As a group, you are asked to get together, discuss your ideas, plan the solutions, compare your solutions, etc. Everyone needs to be able to explain the main ideas in the submission.

2. Please register in groups on blackboard by 9th December. [After 9th December 1 mark will be removed from the overall marks if you have not yet registered in a group, and you may need to complete your coursework alone.] [Note there is no reduction of work, in case you complete your coursework alone.]

3. Marks will be awarded for both correct functionality and good code quality.

4. Please submit exactly one Prolog file. Put your group number and your names and student numbers at the beginning of your file.

5. By submitting this coursework electronically, you state that you fully understand and are complying with the University's policy on Academic Integrity and Academic Misconduct. The policy can be found at www.swansea.ac.uk/academic-services/academic-guide/assessment-issues/academic-integrity-academic-misconduct.

Question 1.

Write the following program and compile it:

```
% Program:  ROYAL

 parent(queenmother,elisabeth).  parent(elisabeth,charles).
 parent(elisabeth,andrew).       parent(elisabeth,anne).
 parent(elisabeth,edward).       parent(diana,william).
 parent(diana,harry).            parent(sarah,beatrice).
 parent(anne,peter).             parent(anne,zara).
 parent(george,elisabeth).       parent(philip,charles).
 parent(philip,andrew).          parent(philip,edward).
 parent(charles,william).        parent(charles,harry).
 parent(andrew,beatrice).        parent(andrew,eugenie).
 parent(mark,peter).             parent(mark,zara).
 parent(william,georgejun).      parent(kate,georgejun).
 parent(william,charlotte).      parent(kate,charlotte).
 parent(philip,anne).            parent(william,louis).
 parent(kate,louis).             parent(harry,archie).
 parent(meghan,archie).
```

The following two predicates define the lists of all female respectively male members of the Royal Family:

```
the_royal_females([queenmother,elisabeth,anne,diana,sarah,beatrice,
                   zara,eugenie,charlotte,kate,meghan]).
the_royal_males([charles,andrew,edward,william,harry,peter,george,
                 philip,mark,georgejun,louis,archie]).
```

Define the following predicates on the persons in the program `ROYAL`.

(1) `the_royal_family/1` (This predicate should hold for a list of all members of the Royal Family. Use the predicates `the_royal_females/1` and `the_royal_males/1` defined above)

(2) `father/2`

(3) `granddad/2`

(4) `has_child/1`.

(5) `ancestor/2`

(6) `sibling/2`

(7) `brother/2`

Translate the following questions into Prolog queries and try them out:

(8) Who is a grandchild of George?

(9) Who has a child (one or more)?

(10) Who is a descendant of Diana?

(11) Who is an ancestor of Archie?

(12) Who has a brother who is granddad?
(Define a predicate `has_brother_who_is_granddad`).

Include your queries as well as your answers as a comment.

[14 marks]

Question 2.

a) Define a predicate `toEven/2` such that `toEven(L1, L2)` takes an input list `L1` of integers and generates an output list `L2` which is `L1` with all odd integers changed into even by doubling each odd number.

b) Write a program `star(N)`, that prints `N` stars, `N-1` stars in the next line and so on. At the end only 1 star is printed. In addition, write a second predicate `star2(N)` that prints `2N-1` lines as follows in the case of `N=3`. [Helper predicates are allowed.]

```
***
**
*
**
***
```

Add your queries as well as the output as a comment.

[8 marks]

Question 3. An example of a recursive predicate and a tail recursive version using an accumulating parameter.

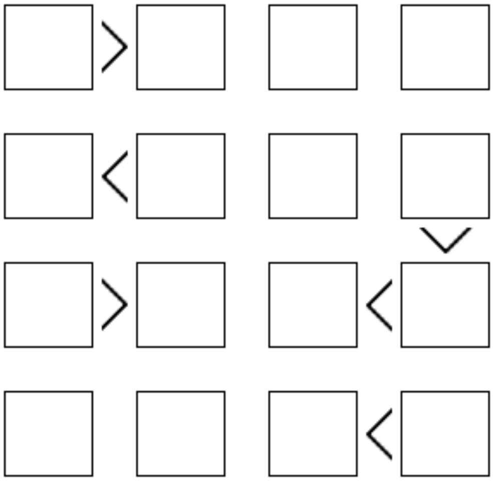
a) The square of the Euclidean distance between two vectors x_i and y_i is $\Sigma_{i=1}^{n}(x_i - y_i) * (x_i - y_i)$. Write a recursive predicate `euclidsqr(X,Y,ED)` which returns the value in `ED` when `X` and `Y` are lists representing vectors of the same length.

b) Next provide a tail recursive solution for the same problem. This function predicate should be called from `euclidsqr2(X,Y,ED)`.

[8 marks]

Question 4. (Backtracking Puzzle)

Here is a Futoshiki puzzle downloaded from the internet (popular in many newspapers).



March 12 - Hard 4 x 4
© Kevin Stone

Figure 1: 4x4 Futoshiki Puzzle

The aim is to place digits 1-4 in the empty cells so that each row and column contains *distinct* digits and the constraints specified by the inequality signs are all satisfied. You are to write a generate and test backtracking program in Prolog to solve this puzzle.

1. Define the predicate `member_rem(E,L,R)` which chooses an element `E` from list `L` leaving remainder `R`.

2. Using the above define `gen_list_n(N,D,L)` which generates a list `L` of `N` *distinct* elements from the list `D` where the length of `D` is ≥ `N`.

```
gen_n(0,_,[]).

gen_n(N,D,[X|Xs]) :-
        N>0,N1 is N-1,
        ....
        gen_n(N1,D1,Xs).
```

Define `gen4(L)` to generate a list of 4 distinct digits from 1-4.

3. To check that two list of numbers `X`, `Y` are different at each entry (i.e, `Xi` differs from `Yi` for all i) define a predicate `distinct_in_entries(X,Y)`.

4. Now you can generate a possible solution `[R1,R2,R3,R4]` where `Ri` are rows of 4 distinct numbers from 1-4 and all the columns consist of distinct numbers as follows: generate `R1` (using `gen4`), then `R2` and check `R1` and `R2` are distinct at all entries; generate `R3` and check this is distinct in entries with `R1` and `R2` and so on. Call this predicate `gen_poss_soln([R1,R2,R3,R4])`.

5. Finally define `solve([R1,R2,R3,R4])` by first generating all possible solutions, and then testing each of the inequalities. Notice that the only constraints in `R1` are on `R11` and `R12` (`R11` > `R12`) so you only need say `R1 = [R11,R12,_,_]`. You should deal with the other rows and constraints in a similar manner.

6. The solver `solve` will find the solution for a 4x4 Futoshiki problem in a reasonable time, but if you scaled it up in an obvious manner for 5x5 problems it would be too inefficient. So produce a more efficient version of `solve` which splits up the generate and test tasks. Call this `solve_in_steps([R1,R2,R3,R4])` and this time, generate `R1` first and test any constraints you can, just involving row 1 (in this case its only `R11 > R12`), then generate R2 and apply the constraints on any variables in `R1` and `R2` and so on. This approach should be able to cope with 5x5 problems (though there are many other improvements you can make to speed up the solver).

[15 marks]

Overall: there will be 5 marks available for code quality, neat presentation, correct submission, registering in a group in time.