Logic Programming
Coursework 1

Available: **October 6, 2014**
Due: **October 20, 2014, 3pm**
(Updated October 11 to correct typo in problem 2)

Submission. Submit your answers in a single file of Prolog source code, which will be tested using Sictus Prolog.

Use the following command on DICE:

```
=>  submit lp 1 <yourFile.pl>
```

This coursework is graded on a scale of 50 points. It counts as 10% of the final grade for LP.

Note: Some exercises rely on material that will not be covered until the programming lecture on October 9.

1. Lists. (Total value **10 points**)

   (a) [**5 points**] A palindrome is a sequence that reads the same forwards and backwards. For example, "`a`", "`aba`", and "`able was i ere i saw elba`" are palindromes. Write a predicate `palindrome(L)` that succeeds when `L` is a palindrome.

   ```
   ?- palindrome([]).
   yes
   ?- palindrome([a,b,b,a]).
   yes
   ?- palindrome([a,b,a,b]).
   no
   ?- palindrome([a,b,X,Y]).
   X = b, Y = a
   ```

   (b) [**5 points**] Write a predicate `allpairs(L,M,N)` that, given lists `L` and `M` as input, succeeds by binding `N` to a list containing all pairs of elements of `L` and `M`. (The term `(X,Y)` builds a pair whose first component is `X` and second component is `Y`.)

   ```
   ?- allpairs([1,2,3],[a,b,c],N)
   N = [(1,a),(1,b),(1,c,),(2,a),(2,b),(2,c),...]
   ?- allpairs([],[1,2,3],N)
   N = []
   ```

2. Aggregation. (Total value **10 points**)

   Consider the following example data about voting in the Scottish independence referendum:

```
indyref(glasgow,194779,364126,75).
indyref(edinburgh,123927,318565,84).
indyref(aberdeen,59390,143484,82).
indyref(stirling,25010,37153,90).
indyref(dundee,53620,93500,78).
```

   Each tuple `indyref(City,For,Votes,Turnout)` lists the percentage voting `For` independence, the total number of `Votes`, and the `Turnout` in a given `City`.

   - [**5 points**] Define a predicate `percentages/1` such that after solving `percentages(L)`, the variable `L` will be bound to a list of pairs `(City,Percentage)` where `City` is a city name and `Percentage` is the percentage of votes for independence (i.e. 100 * `For` divided by `Votes`).
   - [**5 points**] Define a predicate `maxturnout/1` such that after solving `maxturnout(X)`, the variable `X` is bound to the name of the city with the maximum turnout.

   For example:

```
?- percentages(L).
L = [(glasgow,53.49219775572192),...]
?- maxturnout(X).
X = stirling
```

   For full credit, the solution should be independent of the particular example facts above. You may use predicates such as `setof/3`, `bagof/3` or `findall/3`.

3. Logic puzzle. (Total value **10 points**).

   Victor, Wendy, Xavier, Yvette and Zeke all work in the same office building, on five different floors 1–5. None of them works on the same floor. Consider the following constraints:

   - Victor's floor is between Yvette's floor and Zeke's floor.
   - Wendy is not on the first floor.
   - Zeke's floor is two floors above Wendy's.
   - Xavier's floor is not adjacent to Zeke's.

   (a) [**2 points**] Write a predicate `distinct(L)` that tests whether a list of ground terms `L` has no repeats.

   (b) [**3 points**] Write a predicate `generate(V,W,X,Y,Z)` that instantiates the five variable names (representing the five people) with all possible distinct assignments to floors 1–5.

   (c) [**4 points**] Write a predicate `test(V,W,X,Y,Z)` that tests whether the constraints listed above are all satisfied by a given assignment.

   (d) [**1 point**] Include, in a comment in your solution, two solutions to the above constraints generated by running the goal

```
generate(V,W,X,Y,Z), test(V,W,X,Y,Z)
```

4. Flights. (Total value **20 points**) Consider the following facts about costs of flights between different cities:

```
flight(edi,cdg,90).   flight(edi,lhr,50).
flight(lhr,ath,100).  flight(lhr,cdg,70).
flight(cdg,ath,150).  flight(ath,rho,60).
flight(ath,prg,100).  flight(ath,skg,40).
```

   (a) [**1 point**] The above `flight` relation is “asymmetric”: for example, we know that it costs 90 pounds to fly from Edinburgh to Paris (CDG), but not the reverse. Assume that it costs the same to fly from `A` to `B` as it does to fly from `B` to `A`. Write a predicate `flight_sym(A,B,C)` that computes the symmetric closure of `flight`, that is, succeeds if either `flight(A,B,C)` or `flight(B,A,C)` holds.

   (b) [**4 points**] Write a predicate `flight_two_hop(A,B,C)` that succeeds when `A` and `B` are airport codes such that `B` is reachable from `A` in two hops, and binds `C` to the sum of their costs.

   (c) [**10 points**] Write a predicate `reachable(A,B,C)` that, given an airport codes `A` and `B`, succeeds if there is any path from `A` to `B`, binding `C` to the total cost of such a path. Paths should avoid revisiting the same airport and all possile costs should be computed.

   (d) [**5 points**] Write a predicate `cheapest(A,B,C)` that, given airport codes `A` and `B`, succeeds by binding `C` to the cost of the cheapest combination of flights going from `A` to `B`, failing if the two airports are not connected.