Imperial College London – Department of Computing

COMP50007.2: Introduction to Prolog
Lab 3: 'Lists' and 'Arithmetic' *

13 January 2023

Test all the programs that you write for this sheet with representative queries, in each case finding all possible solutions. You could also try tracing the execution of your programs using the debugging tools provided by Sicstus. (See manual.)

Note that Sicstus provides `member/2`, `append/3`, `length/2` as built-in predicates.

Question **??** is more difficult.

1. Write a program `all_members( +X, +Y )` which checks whether all members of a given list `X` are members of another given list `Y`. For example:

```
?- all_members([a,a,d,e], [a,b,c,d,e]).
  yes
?- all_members([a,a,d,e], [a,b,c,d]).
  no
?- all_members([], []).
  yes
```

   The order of the lists, and repetition of members, are irrelevant.

   Use the Sicstus built-in predicate `member/2`.

2. Write a program `pairs( +X, ?Y )` which, given a list `X` of numbers, constructs a list `Y` of the same length such that

   each member of `Y` is `(U, V)` when the corresponding member of `X` is `N` and `U` is `N-1` and `V` is `N+1`.

   e.g., `pairs( [1,3,7], Y )` produces as result `Y = [(0,2),(2,4),(6,8)]`.

3. Write a program `arbpairs( +X, ?Y )` which, given a list `X` of numbers, constructs a list `Y` of the same length such that

   each member of `Y` is `(N, L)` when the corresponding member of `X` is `N` and `L` is either `N` or `2N`.

   e.g., `arbpairs( [2,4], Y )` produces all of the following as possible solutions:

```
Y = [(2,2),(4,4)]
Y = [(2,4),(4,4)]
Y = [(2,2),(4,8)]
Y = [(2,4),(4,8)]
```

*Thanks to Robert Craven, Marek Sergot, Murray Shanahan, and others

4. Write a program `replace_wrap/2`, which replaces every member `X` of a list with `wrap(X)`:

```
?- replace_wrap([a,b,b,c], Res).
  Res = [wrap(a),wrap(b),wrap(b),wrap(c)]
```

   Besides the obvious recursive definition try to define `replace_wrap/2` using `findall/3`.

5. Write a program `even_members( +X, ?Y )` which, given a list `X` constructs the list `Y` consisting of the 2^{nd}, 4^{th}, 6^{th},... members of the input list `X`:

```
?- even_members([a,b,c,d,e,f,g], Res).
  Res = [b,d,f]
```

6. Generalise the previous question. Write a program `odd_even_members( +X, ?Y, ?Z )` which, given a list `X` constructs the lists `Y` and `Z` consisting of the 1^{st}, 3^{rd}, 5^{th},... and 2^{nd}, 4^{th}, 6^{th},... members of the input list `X`, respectively:

```
?- odd_even_members([a,b,c,d,e,f,g], Odds, Evens).
  Evens = [b,d,f], Odds = [a,c,e,g]
```

7. For the purposes of this exercise, suppose that an arithmetic term is defined to be

	a `number`	
else	a term `a(X,Y)`	where `X` and `Y` are both arithmetic terms
else	a term `m(X,Y)`	where `X` and `Y` are both arithmetic terms

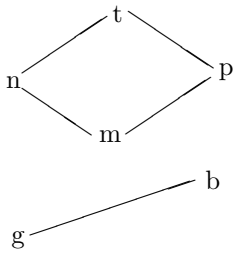
   where term `a(X,Y)` represents $\texttt{X} + \texttt{Y}$, whilst `m(X,Y)` represents $\texttt{X} * \texttt{Y}$.

   Write a program `numval( +A, ?V )` which, given the arithmetic term `A`, computes the numerical value `V` that it represents.

   For example, if `A = a(m(3,a(2,5)),a(2,3))` then `A` represents $(3 * (2 + 5)) + (2 + 3)$ and so `V = 26`.

   The only predicates that you need are `numval/2` itself, `number/1` and `is/2`.

8. *More challenging:* Given a graph such as:



write a program `connected_parts/1` to determine the set of connected components of the graph. A connected component is itself a set of nodes, such that every two distinct nodes in it are connected via some sequence of edges. For example, in the given graph there are two connected components, so you would expect as result `[ [b, g], [m, n, p, t] ]`. Note that all sets in the resulting list have their elements sorted.

You will need to decide how to represent the graph in terms of Prolog clauses. Use `node/1` facts to represent nodes, and `edge/2` facts to represent edges. This allows for isolated nodes in the graph. (In exercise sheet 2, graphs were represented using `arc/2` facts. There is no particular reason to pick `edge/2` this time.)

Use the following method. Keep an (ordered) list of connected components constructed so far. (Start by putting all the nodes into their own singleton lists. In the example above this would be the list `[ [b], [g], [m], [n], [p], [t] ]`. These singleton lists are obviously all connected.) Look for any pair of components that can be joined into one. Two components can be joined into one if they have elements with an edge between them. Merge these two components keeping the nodes in sorted order. The procedure terminates when no more merging/joining of components can take place.

The first step is to construct the initial set of singleton lists, one for each node. You can do this using `findall`. The top level of the `connected_parts/1` is as follows:

```
connected_parts(Components) :-
   findall([N], node(N), Initial),
   join_components(Initial, Components).
```

Now write the (recursive) `join_components/2` program.