University of Toronto
CSC324 –Programming Languages, winter 2011

Assignment # 1

Details
- Topics: Prolog, logic oriented programming.
- Weight: 15%
- Due Date: this assignment is due on Feb 6th, 2011 @ 11:59pm
- This assignment is to be done individually.

Note
You may use helper relations as needed. However, YOU MAY NOT USE ANY OF THE FOLLOWING:

; ("or")
-> ("if-then")
! (cut)

or any other special operators in Prolog (which generally subvert the pure logic programming paradigm). If you are in doubt about what's allowed, ask.

Marking
- Your code will be marked for correctness (80%) and code documentation & style (20%). Check page 4 of this handout for Documentation and Style guidelines.
- Since we plan to test your code using an auto-marker, you must use the exact function names/arguments as specified and use the same file names and folder name as specified.
- You will lose 25% of the total mark if you use wrong file names or predicate names.
- You will lose 10% of the total mark if your submission fails to load.

Submission
Place all your files in a folder named after your UTORID and submit that folder zipped through the portal (if your UTORID is abcd, the file should be named abcd.zip). A link for submission is provided in the assignments folder in the portal (http://portal.utoronto.ca)

Prolog Notational Conventions: Predicate and Mode Spec
We shall use the following notation when referring to Prolog predicates: Predicates in Prolog are distinguished by their name and their arity. The notation name/arity is therefore used when it is necessary to refer to a predicate unambiguously; e.g. append/3 specifies the predicate which is named "append" and which takes 3 arguments. Sometimes, we may be interested in specifying how specific predicates are mean to be used. To that end, we shall present a predicate's usage with a mode spec which has the form: name(arg, ..., arg) where each arg denotes how that argument should be instantiated in goals, and has one of the following forms:

+ArgName This argument should be instantiated to a non-variable term.
-ArgName This argument should be un-instantiated.
?ArgName This argument may or may not be instantiated.

For example delete(+List,?Elem,?NewList) states that, when using delete/3, the first argument should be instantiated whereas the second and third arguments may or may not be instantiated.

1. [*10 points*] Write a predicate **abs(+X,?Y)** which takes a number X as input and computes the absolute value |X| as output. *Sample invocations:*

```
| ?- abs(0,0).
yes

| ?- abs(-1,1).
yes

| ?- abs(-1,Y).
Y = 1

| ?- abs(5,Y).
Y = 5
```

Write your answer in a file called **abs.pl**

2. [*20 points*] Write a Prolog predicate **seq(+First, +Last, -N)** that instantiates N to each integer value from First through Last, inclusive. First and/or Last may be negative. Assume that First and Last are integers. *Sample invocation:*

```
| ?- seq(1, 3, X).
X = 1 ;
X = 2 ;
X = 3 ;
no

| ?- seq(-3, 2, X).
X = -3 ;
X = -2 ;
X = -1 ;
X = 0 ;
X = 1 ;
X = 2 ;
no

| ?- seq(1,1,X).
X = 1 ;
no
| ?- seq(2,-2,X).
no
```

Write your answer in a file called **seq.pl**

3. [*30 points*] Write a prolog predicate **iprod**(+List1,+List2,-Result) that calculates the inner product (or dot product) of two vectors a and b . List1 and List2 are both fully instantiated flat lists of integers and are both of the same length. *Sample invocation:*

```
| ?- iprod([1,2,4],[3,5,6],Result).
Result=37                                   % 1*3+2*5+4*6
```

Your predicates should be submitted in a single file named **iprod.pl**

4. [*40 points*] Assume that you have been hired to work on a facebook like website! The friends information is stored in a file called friends.pl. Here is sample file content:

```
friend(christian,margaret).
friend(christian,jas).
friend(christian,todd).
friend(christian,ji).
friend(christian,geener).
friend(todd,christian).
friend(todd,susan).
friend(susan,todd).
friend(jas,christian).
friend(jas,geener).
friend(jas,clark).
friend(geener,christian).
friend(geener,jas).
friend(geener,ji).
friend(clark,pat).
friend(pat,mike).
friend(pat,clark).
friend(margaret,christian).
friend(ji,christian).
friend(ji,geener).
```

You have been asked to write a predicate

friendster(+ID1,+ID2,-Path,+MaxPathLength)

which will find if user ID1 is connected to user ID2 through some friend. Path will be assigned the list of friends along one path. Two friends might be connected through more than one path. Each ID is unique. Friend(ID1,ID2) implies Friend(ID2,ID1). MaxPathLength indicates the maximum number of friends in the list including ID1 and ID2. *Sample invocation:*

```
| ?- friendster(christian,clark,L,3).
L = [christian,jas,clark]
```

```
| ?- friendster(christian, susan,L,3).
L = [christian, todd, susan] ;
no

| ?- friendster(christian,mike,L,3).
no

| ?- friendster(christian,mark,L,3).
no


| ?- friendster(christian,christian,L,3).
no

| ?- friendster(christian,geener,L,3).
L = [christian,geener] ;
L = [christian,jas,geener] ;
L = [christian,ji,geener] ;
no
```

Note that the path from christian to mike is too long for them to be friends, and that christian is friends with geener through three different paths. *Hint: one possible solution involves using a helper function so that you can keep track of what people have already been visited.*

Write your answer in a file called **friendster.pl** (Note: do not add a statement to load friends.pl in your answer. When marking, the auto-marker will do that).

Documentation && Style

A. Begin each clause on a new line and indent all but the first line of each clause. E.g.:

```
same_length([], []).
same_length([_|L1], [_|L2]) :-
        same_length(L1, L2).
```

B. Put each sub-goal on a separate line. E.g.:

```
ord_union_all(N, Sets0, Union, Sets) :-
        A is N/2,
        Z is N-A,
        ord_union_all(A, Sets0, X, Sets1),
        ord_union_all(Z, Sets1, Y, Sets),
        ord_union(X, Y, Union).
```

C. Use layout to make comments more readable. E.g.:

```
% This predicate classifies C as:
%   - whitespace (ASCII < 33);
%   - alphabetic (a-z, A-Z);
%   - numeric (0-9); or
%   - symbolic (all other characters).
```

D. Avoid comments to the right of the code!

E. If a predicate represents a property or relation, its name should be a noun, noun phrase, adjective, prepositional phrase, or indicative verb phrase. E.g.

```
sorted_list, well_formed_tree, parent (nouns or noun phrases)
well_formed, ascending (adjectives)
in_tree, between_limits (prepositional phrases)
contains_duplicates, has_sublists (indicative verb phrases).
```

F. Each of your files should follow the following layout

```
% AUTHOR:
% UTORID:
%
% FACTS


% RULES
```

G. Make sure to document every rule.