CS 335 - Homework 6

Deadline:

Due by 11:59 pm on Monday, April 4 (unusual time due to late posting AND Cesar Chavez holiday on Thursday, March 31st)

How to submit:

Submit your files for this homework using `~st10/335submit` on nrs-labs, with a homework number of `6`

Purpose:

To work with lists and arithmetic in Prolog, and to continue development of your Prolog game.

Important notes:

- You are expected to use SWI-Prolog for this assignment.

The Problems:

Problem 1 - 5 points

We discussed the `member/2` predicate in class. Now also consider the `append/3` predicate:

```
% the idea here is that append(List1, List2, ResultList). should
%    be true if the result of appending List1 and List2 is ResultList

append([], List, List).

append([Head1|Tail1], List2, [Head1|Tail2])
    :- append(Tail1, List2, Tail2).
```

In addition to `member/2` and `append/3`, another useful built-in predicate in Prolog is `reverse(List1, List2)`, which is true if `List1` and `List2` are both lists, and `List2` contains the same elements as `List1`, except in reverse order.

(If you look at its listing, you'll see a different approach to a recursive list predicate -- `reverse/2` uses an auxiliary predicate, `reverse/4`.)

You don't have to write `reverse`, though -- you just need to *use* it to help write a much-simpler predicate `palindrome(List)`, which is true if `List` is a palindrome (for our purposes here, a list whose elements read the same forwards and backwards; for example,

```
?- palindrome([s,a,i,p,p,u,a,k,a,u,p,p,i,a,s]).
```

...should be true, and

```
?- palindrome([h, a, n, n, a, h]).
```

...should also be true.

Start a Prolog knowledge base `hw6-1.pl`. The first thing in this file should be comments containing at least:

- your name
- `CS 335 - Homework 6 - Problem 1`
- and the last-modified date

Then include your predicate `palindrome/1`. Use `protocol` to create a record of a `swipl` session in a file named `hw6-1-ex.txt` in which you:

- load your `hw6-1.pl` knowledge base;
- test your `palindrome/1` predicate on at least the following queries:
  - the above examples,
  - an example that fails,
  - another example that succeeds,
  - and as many additional examples as you would like.

Submit your resulting `hw6-1.pl` and `hw6-1-ex.txt`.

Problem 2 - 10 points

(adapted from UCF's COT 4020 Homework 3, Spring 2007)

Suppose we are given a knowledge base with the following facts:

```
translation(un, one).
translation(deux, two).
translation(trois, three).
translation(quatre, four).
translation(cinq, five).
translation(six, six).
translation(sept, seven).
translation(huit, eight).
translation(neuf, nine).
translation(dix, ten).
```

Start a Prolog knowledge base `hw6-2.pl`. The first thing in this file should be comments containing at least:

- your name
- `CS 335 - Homework 6 - Problem 2`
- and the last-modified date

Then include the above facts along with a predicate `listE2F(EngList,FrList)` which can use

the above facts to help translate a list of English number words to the corresponding list of French number words or vice versa. (Or, that is true if EngList is a list of English number words, and FrList is a list of the corresponding French number words.) For example:

```
?- listE2F(EngNums,[un, neuf, deux]).
```

...should be true with the unification:

```
EngNums = [one, nine, two].
```

(It is OK if, when you enter `;` after the above unification, you get `false`:

```
?- listE2F(EngNums,[un, neuf, deux]).
EngNums = [one, nine, two] ;
false.
)
```

Your predicate should also work in the other direction. For example, if you give it the query

```
?- listE2F([one, seven, six, two], FrNums).
```

...then this should be true with the unification:

```
FrNums = [un, sept, six, deux].
```

Hint: to write this predicate, first ask yourself `How do I translate the empty list of number words?'. That's the base case. For non-empty lists, first translate the head of the list, then use recursion to translate the tail.

Use `protocol` to create a record of a `swipl` session in a file `hw6-2-ex.txt` in which you:

- load your `hw6-2.pl` knowledge base,
- run the above 2 queries, and
- also test your `listE2F/2` predicate on at least 3 additional queries of your choice that you think show something interesting.

Submit your resulting `hw6-2.pl` and `hw6-2-ex.txt`.

Problem 3 - 10 points

Start a Prolog knowledge base `hw6-3.pl`. The first thing in this file should be comments containing at least:

- your name
- `CS 335 - Homework 6 - Problem 3`
- and the last-modified date

Then write a predicate `longer(List1, List2)` that is true if `List1` is longer (has more top-level elements) than `List2`.

For example,

```
?- longer([a, b, c], [1, [1, 2, 4]]).
```

...should be true because the first list has three elements and the second list has just two (top-level)

elements.

Use `protocol` to create a record of a swipl session in a file `hw6-3-ex.txt` in which you:

- load your `hw6-3.pl` knowledge base,
- test your `longer/2` predicate on at least the following queries:
  - the above example,
  - an example for each base case,
  - an example that fails,
  - another example that succeeds,
  - and as many additional examples as you would like.

Submit your resulting `hw6-3.pl` and `hw6-3-ex.txt`.

Problem 4 - 10 points

Start a Prolog knowledge base `hw6-4.pl`. The first thing in this file should be comments containing at least:

- your name
- `CS 335 - Homework 6 - Problem 4`
- and the last-modified date

Then write a `remove/3` predicate. `remove(Item, List1, List2)` should be true when `List2` is a list whose contents are those of `List1` with all 'top-level' instances of `Item` removed. That is,

```
remove(3, [4, 3, 5, 3, 6, 3], ResultList).
```

...should be true with the unification:

```
ResultList = [4, 5, 6]
```

And,

```
remove(3, [4, 5, 6], ThisResult).
```

...should be true with the unification:

```
ThisResult = [4, 5, 6]
```

Hints:

- what should be resulting list if you try to remove an item from the empty list?
- remember that `X==Y` is true if `X` and `Y` are unified to the same value;
- remember that negation is `\+` (or the more-readable predicate `not/1`, if you prefer)

Use `protocol` to create a record of a `swipl` session in a file `hw6-4-ex.txt` in which you:

- load your `hw6-4.pl` knowledge base,

- test your `remove/3` predicate on at least the following queries:
  - the above two examples,
  - an example for each base case,
  - an example that fails,
  - another example that succeeds,
  - and as many additional examples as you would like.

Submit your resulting `hw6-4.pl` and `hw6-4-ex.txt`.

Problem 5 - 10 points

Start a Prolog knowledge base `hw6-5.pl`. The first thing in this file should be comments containing at least:

- your name
- `CS 335 - Homework 6 - Problem 5`
- and the last-modified date

We discussed the `member/2` predicate in lecture; recall that it is built-in in `swipl`, and you can use the query:

```
?- listing(member/2).
```

...to remind yourself how it it works.

`member/2` is provided by `swipl` because it is a very handy predicate to have for use in other predicates -- for example, you should be able to use it to good advantage in writing the following predicate.

Now, in knowledge base `hw6-5.pl`, write a predicate `subset(Subset, Set)` that should be true when list `Subset` is indeed a subset of list `Set` (that is, when all of the elements in list `Subset` are also in list `Set`, and the order of the elements doesn't matter).

That is,

```
subset([8, 1, 3], [2, 1, 5, 4, 8, 3, 5]).
```

...should be true, and

```
subset([2, 4, 7], [7, 4, 2]).
```

...should be true, and

```
subset([2, 4], [5, 3, 1, 2]).
```

...should be false.

Use `protocol` to create a record of a `swipl` session in a file `hw6-5-ex.txt` in which you:

- load your `hw6-5.pl` knowledge base,
- test your `subset/2` predicate on at least the following queries:

  - the above three examples,
  - an example for each base case,
  - an example that fails,
  - another example that succeeds,
  - and as many additional examples as you would like.

Submit your resulting `hw6-5.pl` and `hw6-5-ex.txt`.

Problem 6 - 10 points

Start a Prolog knowledge base `hw6-6.pl`. The first thing in this file should be comments containing at least:

- your name
- `CS 335 - Homework 6 - Problem 6`
- and the last-modified date

As a little practice with arithmetic in Prolog, now write a predicate `sum(NumList, Sum)` that should be true when the sum of the numbers in `NumList` are equal to `Sum`.

That is,

```
sum([], 0).
```

...should be true, and

```
sum([2, 4, 7], 13).
```

...should be true, and

```
sum([2, 4], 7).
```

...should be false.

Use `protocol` to create a record of a `swipl` session in a file `hw6-6-ex.txt` in which you:

- load your `hw6-6.pl` knowledge base,
- test your `sum/2` predicate on at least the following queries:
  - the above three examples,
  - another example that succeeds,
  - and as many additional examples as you would like.

Submit your resulting `hw6-6.pl` and `hw6-6-ex.txt`.

Problem 7 - 45 points

Keep working on your Prolog adventure game that you started in Homework 5, but now there are a couple of **additional** requirements:

  - you must use a **list** somewhere in your game;
  - you must use **arithmetic** somewhere in your game.

By this homework's (Homework 6's) deadline, you will be expected to submit working attempts at at least **THREE** of the requirements locked-door/hidden-object/incomplete-object/limited-resources, and you must have begun to make use of at least one list and at least one use of arithmetic in your game. (If the list and arithmetic features are not yet completed, but are started, that is acceptable at this stage.)

You need to submit (for Homework 6 Problem 7):

- your knowledge base thus-far (in a file named `hw6-game2.pl`),
- a **transcript** of a sample run of this latest version of your game (in a file named `hw6-game2-sample.txt` created by the `protocol` predicate)
- a `hw6-game2-readme.txt` file that:
  - briefly **describes** your game,
  - briefly **describes** your created-so-far locked-door/hidden-object/incomplete-object/limited-resource,
  - briefly **describes** how you are using lists in your game,
  - briefly **describes** how you are using arithmetic in your game.