Spring, 1992 Name: ______________________

CS 342 — Principles of Programming Languages

Prolog Test (Problem Solving, Version 2)

Before you begin, please take a moment to look over the entire test so that you can budget your time. The usual conventions from the Course Specification for problem solving tests apply.
A special rule is that *your programs only have to be able to answer queries of the form given in the examples for each problem.* If there are no queries that tell you that you have to be able to run the program "backwards," then you don't have to be able to run it backwards.
In your solutions you may use the following facts and rules from the textbook without further mention. Any other facts or rules that you wish to use must be written in your solution. If you wish to use facts or rules from your solutions in other parts of the exam, you must clearly state where we can find those. Remember that you can define "helping" predicates if you wish. You may also use simplify to convert from difference lists and back again unless the problem specifically prohibits that.

```
(infer (member X (cons X L)))
(infer (member X (cons Y M)) from (member X M))
(infer (addtoend nil X (cons X nil)))
(infer (addtoend (cons Y L) X (cons Y M)) from (addtoend L X M))
(infer (reverse nil nil))
(infer (reverse (cons X L) M) from (reverse L N) (addtoend N X M))
(infer (append nil L L))
(infer (append (cons X L) M (cons X N)) from (append L M N))
(infer (member X L) from (append L1 (cons X L2) L)).
(infer (lookup K A L) from (member (pair K A) L))
(infer (mult 0 Y 0))
(infer (mult X Y Z) from (minus X 1 V) (mult V Y W) (plus W Y Z))
(infer (fac 0 1))
(infer (fac N R) from (minus N 1 N1) (fac N1 R1) (mult R1 N R))
(infer (naive-sort L M) from (permutation L M) (ordered M))
(infer (<= X X))
(infer (<= X Y) from (less X Y))
(infer (ordered nil))
(infer (ordered (cons A nil)))
(infer (ordered (cons A (cons B L))) from (<= A B) (ordered (cons B L)))
(infer (permutation nil nil))
(infer (permutation L (cons H T))
      from (append V (cons H U) L) (append V U W) (permutation W T))
(infer (partition H (cons A X) (cons A Y) Z) from (<= A H) (partition H X Y Z))
(infer (partition H (cons A X) Y (cons A Z)) from (less H A) (partition H X Y Z))
(infer (partition H nil nil nil))
(infer (quicksort nil nil))
(infer (quicksort (cons H T) S)
      from
      (partition H T A B)
             (quicksort A A1)
             (quicksort B B1)
             (append A1 (cons H B1) S))
(infer (simplify (diff X X) nil))
(infer (simplify (diff (cons X Y) Z) (cons X W)) from (simplify (diff Y Z) W))
(infer (diffappend (diff L X) (diff X Y) (diff L Y))).
```

1. (5 points) Write a predicate **not-equal** that takes two integers and is satisfied if and only if its first argument is different from its second. For example,

```
-> (infer? (not-equal 3 4))
Satisfied

-> (infer? (not-equal 27 7))
Satisfied

-> (infer? (not-equal 3 3))
Not satisfied
```

2. (10 points) Write a predicate `remove-duplicates`, such that `(remove-duplicates L1 L2)`, holds if the list of integers `L2` has the elements of the list of integers `L1` in some (unspecified) order, but without duplicates. For example, the query

```
(infer? (remove-duplicates (cons 1 (cons 2 (cons 1 (cons 2 nil)))) Lst)
        (print Lst))
```

should print either `(cons 1 (cons 2 nil))` or `(cons 2 (cons 1 nil))`. The choice gives you flexibility in your coding; thus only one of the following queries needs to be satisfied by your code; the other may fail to be satisfied.

```
(infer? (remove-duplicates (cons 3 (cons 4 (cons 4 (cons 3 nil)))) (cons 3 (cons 4 nil))))
(infer? (remove-duplicates (cons 3 (cons 4 (cons 4 (cons 3 nil)))) (cons 4 (cons 3 nil))))
```

Note that the query

```
(infer? (remove-duplicates (cons 3 (cons 4 (cons 4 nil))) (cons 3 (cons 4 (cons 4 nil)))))
```

must fail. This only has to work for the above kinds of examples and for integer lists.

3. (5 points) Suppose that the predicate `is-variable` were built-in to our version of Prolog. For example,

```
-> (infer? (is-variable X))
Satisfied

-> (infer? (is-variable instantiated))
Not Satisfied

-> (infer? (is-variable 1))
Not Satisfied
```

Specify the relation `is-difference-list` such that `(is-difference-list` *l*`)` is satisfied if and only if *l* is a difference list. For example,

```
-> (infer? (is-difference-list (diff (cons 1 X) X)))
Satisfied

-> (infer? (is-difference-list (diff (cons 1 nil) nil)))
Not Satisfied
```

You may use `is-variable` in your solution.

4. (7 points) Suppose we write a list of alternatives using *or-lists*, where an or-list is a Prolog term that satisfies the following predicate

```
(infer (is-or-list (or A true)))
(infer (is-or-list (or Hd Tail))
  from (is-or-list Tail))
```

So, for example

```
(or bush true)
(or (+ A B) (or (* A B) (or (- A B) (or (/ A B) true))))
```

are both or-lists.

Write a predicate `one-of` such that `(one-of` *ol t*`)` holds if t can be unified with one of the elements in the or-list ol. You can assume that t is *not* an or-list itself, and that the first argument to `one-of` is a well-formed or-list. For example,

```
-> (infer? (one-of (or 1 (or 2 true) 2))
Satisfied

-> (infer? (one-of (or (+ A B) (or (* A B) (or (- A B) (or (/ A B) true)))) (* 1 2)))
Satisfied
```