Spring, 1992 Name: _____

CS 541 — Programming Languages 1

Logic Programming Test

(Problem Solving, Version 1)

Before you begin, please take a moment to look over the entire test so that you can budget your time. Each numbered problem has equal weight.
If you get an incomplete on this test, you can get at most a 'B' on the makeup test.
For programming problems, there is usually more than one right answer, hence it is important to present with your program evidence that it is correct. Answers to programming questions will be scored on the following points:

- elegance and clarity (conciseness, using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution, and how clear and convincing your argument is that the code is correct)
- correctness (whether your program solves the given problem correctly in the given programming language)

You should try to eliminate all syntax errors from your programs, as they will affect both the correctness and clarity of your solution; you may receive no points for programs with major syntax errors. Run-time space and time efficiency of programs is not important, unless the problem says so. Also, checking of external inputs, prompting, labeling output, and other conventions for dealing with humans are not important (for this class), unless the problem says so. The breakdown of points between these categories is roughly as follows.

- 60% for elegance and clarity (using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution),
- 40% for correctness (whether your program solves the given problem correctly in the given subset of the given language)

You are encouraged to use auxiliary subroutines whenever they are helpful.
For this test, you may use any of the following built-in predicates in Prolog:

```
+, -, *, /, is, =:=, <, >, =<, >=, =, true, !, not, fail
```

If you wish to use functions from your solutions in other parts of the exam, you must clearly state where (which problem) we can find those function definitions.

1. The (abstract) syntax of expressions in Smalltalk may be described by the following rules in Prolog.

```
identifier(id(X)).
expression(E) :- identifier(E).
expression(block(Name,Body)) :- identifier(Name), expression(Body).
expression(send(Obj,Msg)) :- expression(Obj), message(Msg).
message(msg(Name,Args)) :- list_of_expressions(Args).
list_of_expressions([]).
list_of_expressions([Head|Tail]) :-
    expression(Head),
    list_of_expressions(Tail).
```

Using the above definitions, translate the following English statements into Prolog. For example, the translation of: "An identifier X is free in an identifier expression id(Y) if X is the same as Y" would be

```
free_in(id(X),id(X)).
```

or some equivalent form.

(a) An identifier X is free in a list of expressions if it is free in any element of the list.

(b) An identifier X is free in a send expression send(Obj,Msg) if X is free in either Obj or in Msg.

(c) An identifier X is free in a block expression block(Name,Body) if X not the same as Name and X is free in Body.

2. Consider the natural numbers in unary notation, defined by the following. (The functor `s` stands for "successor", hence 3 is written `s(s(s(0)))`.

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
le(0,X) :- natural_number(X).
le(s(X),s(Y)) :- le(X,Y).
lt(0,s(X)) :- natural_number(X).
lt(s(X),s(Y)) :- lt(X,Y).
plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

   Specify the relation `remainder`/3, such that if X, Y, and Z are natural numbers, `remainder(`X`,`Y`,`Z`)` holds if Z is the remainder of dividing X by Y. You may assume that Y is non-zero. For example, `remainder(s(s(s(s(s(0))))),s(s(s(0))),s(s(0)))` should hold, as the remainder of 5 divided by 3 is 2. Note also that the remainder of 8 divided by 3 is 2. Finally, `remainder(s(s(s(0))),s(s(s(s(s(0))))),s(s(s(0))))` holds because the remainder of 3 divided by 5 is 3. Your program must be able to answer queries such as:

```
?- remainder(s(s(s(s(s(0))))),s(s(s(0))),R).
```

   and have Prolog find a value for `R`.

3. Specify the relation `remove_duplicates/2`, such that `remove_duplicates(L1,L2)`, holds if the list `L2` has the elements of the list `L1` in some (unspecified) order, but without duplicates. For example: either `remove_duplicates([1,2,1,2],[1,2])` or `remove_duplicates([1,2,1,2],[2,1])` should hold (or both), but it is okay to have only one of them succeed. However, `remove_duplicates([1,2,1,2],[1,2,1,2])` should fail. Your program must be able to answer queries such as:

   ```
   ?- remove_duplicates([1,2,3,1,3,1], S).
   ```

   and have Prolog find a value for `S`.

4. Specify the relation `intersection/3`, such that `intersection(L1,L2,L3)` holds if the only elements in the list `L3`, are elements in the intersection of the lists `L1` and `L2`. The order of the list `L3` is left unspecified, your program must succeed with one order, but not necessarily with all orders; similarly, your program may either allow or disallow duplicates in `L3`. For example, either `intersection([1,2,3],[4,3,1],[1,3])` or `intersection([1,2,3],[4,3,1],[3,1])` should succeed. However, `intersection([1,2,3],[4,3,1],[1,2,3,4])` should fail. Your program should be able to be queried as follows:

   ```
   ?- intersection([1,2,3], [4,3,1], S).
   ```

   and have Prolog find a value for `S`.