COMP 410
Fall 2020
Final Practice Exam

The topics on this practice exam reflect **ONLY** those which have been covered since the last exam. The real final is **CUMULATIVE**, so it will include questions similar to the previous practice exams. However, the final will be biased towards the sort of questions below.

Prolog Metainterpreters

1.) Write a metainterpreter which shows the number of conjunctions which were needed to compute a particular solution. Example queries follow. Your metainterpreter needs to handle only the rules necessary to execute these queries below.

```
?- interpret((X is 1 + 1, Y is 2 + 2), ConjunctionCount).
X = 2, Y = 4, ConjunctionCount = 1.

% This definition is used in the query below
% myLength([], 0).
% myLength([_|T], Len) :-
%    myLength(T, TLen),
%    Len is TLen + 1.

?- interpret(myLength([a, b, c, d], Len), ConjunctionCount).
Len = 4, ConjunctionCount = 4.
```

```
interpret(true, 0) :- !.
interpret(is(A, B), 0) :-
    !,
    is(A, B).
interpret((A, B), FinalCount) :-
    !,
    interpret(A, ACount),
    interpret(B, BCount),
    FinalCount is ACount + BCount + 1.
interpret(Call, Count) :-
    clause(Call, Body),
    interpret(Body, Count).
```

Constraint Logic Programming and Peano Arithmetic

2.) Using CLP constraints, write a query which finds all integers `X` and `Y` such that:

```
X >= 0
X <= 10
Y >= 0
Y <= 10
X + Y < 10
```

```
?- X #>= 0,
   X #=< 10,
   Y #>= 0,
   Y #=< 10,
   X + Y #< 10,
   label([X, Y]).
```

3.) Via the Peano axioms, we can define natural numbers `n` as follows:

```
n ::= zero | succ(n)
```

...where `succ(n)` represents the successor to some other natural number `n`.

3.a.) Write out 5 as a natural number encoded with the Peano axioms.

```
succ(succ(succ(succ(succ(zero)))))
```

3.b.) Assume the presence of a procedure `add/3`, which takes three natural numbers encoded with the Peano axioms. The first two arguments are inputs, and the third argument is the sum of the two inputs. Define a procedure `multiply/3`, which takes three natural numbers encoded with the Peano axioms. `multiply/3` multiplies the first two arguments together, placing the result in the third argument. You may assume the first two inputs will always be provided. As a hint:

- `0 * n = 0`
- `1 * n = n`
- `n * m = m + ((n - 1) * m)` for `n, m > 1`

```
multiply(zero, _, zero) :- !.
multiply(_, zero, zero) :- !.
multiply(succ(zero), N, N) :- !.
multiply(N, succ(zero), N) :- !.
multiply(succ(N), M, Result) :-
    multiply(N, M, Rest),
    add(M, Rest, Result).
```