**COMP 410**
**Fall 2021**

## Prolog Metainterpreters

Consider the definition of list append (named `myAppend`) shown below, which we will make use of for example queries:

```
myAppend([], List, List).
myAppend([H|T], List, [H|Rest]) :-
    myAppend(T, List, Rest).
```

1.) Write a procedure named `interp0`, which acts as a metainterpreter that can handle `myAppend`. As a hint, this metainterpreter needs only support for true, conjunction, and calls. An example query is shown below:

```
?- interp0(myAppend([1, 2, 3], [4, 5], List)).
List = [1, 2, 3, 4, 5].
```

2.) Write a procedure named `interp1`, which will print out exactly what is called during the course of execution. As a hint, `writeln` will print out a Prolog term, followed by a newline. Most of this code should be identical to that of `interp0`; only how calls are handled needs to be changed. An example query is shown below, complete with output showing different calls made during execution of `myAppend`:

```
?- interp1(myAppend([1, 2, 3], [4, 5], List)).
myAppend([1,2,3],[4,5],_G1634)
myAppend([2,3],[4,5],_G1730)
myAppend([3],[4,5],_G1747)
myAppend([],[4,5],_G1764)
List = [1, 2, 3, 4, 5].
```

3.) Write a procedure named `interp2`, which works like `interp1`, *but* it also prints out the result of calls. This should look much like `interp1`, and only the rule for calls should change. An example query is shown below:

```
?- interp2(myAppend([1, 2, 3], [4, 5], List)).
Call: myAppend([1,2,3],[4,5],_G1634)
Call: myAppend([2,3],[4,5],_G1730)
Call: myAppend([3],[4,5],_G1747)
Call: myAppend([],[4,5],_G1764)
Return: myAppend([],[4,5],[4,5])
Return: myAppend([3],[4,5],[3,4,5])
Return: myAppend([2,3],[4,5],[2,3,4,5])
Return: myAppend([1,2,3],[4,5],[1,2,3,4,5])
List = [1, 2, 3, 4, 5].
```

4.) Write a procedure named `interp3`, which works like `interp2` but it also displays how deep the call stack is at any given moment. Perhaps the easiest way to track call stack depth is to implement a helper which tracks the call stack depth. With this in mind, unlike with `interp1` and `interp2`, this will likely require you to change all rules slightly to forward along and update the call stack depth as appropriate. An example query is shown below (which looks a lot like the output of `trace`, which is no accident):

```
?- interp3(myAppend([1, 2, 3], [4, 5], List)).
Call (0): myAppend([1,2,3],[4,5],_G1634)
Call (1): myAppend([2,3],[4,5],_G1736)
Call (2): myAppend([3],[4,5],_G1759)
Call (3): myAppend([],[4,5],_G1782)
Exit (3): myAppend([],[4,5],[4,5])
Exit (2): myAppend([3],[4,5],[3,4,5])
Exit (1): myAppend([2,3],[4,5],[2,3,4,5])
Exit (0): myAppend([1,2,3],[4,5],[1,2,3,4,5])
List = [1, 2, 3, 4, 5].
```