Programming Languages
Homework 6
Due Wednesday, February 19th at 2 AM

For this and next homework, we are writing in Prolog. You can use either swipl on openlab or the web app for prolog here to create a program. For the web app, download your program to your computer or save it in the cloud. For using swipl on openlab, refer to the guide here.

If you would like an example Prolog file to look over, you can refer to Dr. Klefstad's example file in openlab with this command:

```
$ cat ~klefstad/public_html/public/141/hw6/mydefs
```

Implement the following relations in Prolog. Where applicable, run them backward.

1. **[10]** Write my_reverse which does the obvious.

   ```
   ?- my_reverse([a,b,c,d], R).
   R = [d,c,b,a]
   ```

2. **[10]** Write my_length which relates the list to its length.

   ```
   ?- my_length([[a,b],[c,d],[e,f]], Y).
   Y = 3
   ```

3. **[10]** Write my_subset that takes a relation and a list. The last parameter is those in the list that satisfy the relation. You will need operator =.. for my_subset (read this). It is an equal followed by two dots.

   ```
   ?- my_subset(atom, [a,[b],c], Y).
   Y = [a,c]
   ```

4. **[10]** Write my_member which relates an element to a list that the list is in.

   ```
   ?- my_member(a,[a,b,c]).
   Yes
   ?- my_member(Y,[a,b,c]).
   Y = a;
   Y = b;
   ```

```
Y = c;
No
```

5. **[10]** Write my_intersect that relates a list to another list. The third list is the subset of elements that are in both the first two lists.

```
?- my_intersect([a,b,c],[[c],d,b,e,a], R).
R = [a,b]
```

Implement the following Prolog relations. If applicable, the relation should be able to respond to the backward query.

NOTE: In writing the following Prolog predicates, you may use only these primitive functions:

- append
- member
- not
- predicates you wrote above

6. **[20]** Define the predicate compute_change(Money, Quarter, Dime, Nickle, Penny). Try to minimize the number of coins you should give for the first answer. (No need to test backward query, but run several tests similar to this one). You will need to use the operator "is" for this one to do an evaluation of arithmetic.

```
?- compute_change(33,Q,D,N,P).
Q = 1 D = 0 N = 1 P = 3
```

7. **[15]** Define the predicate compose(L1, L2, L3) such that L3 consists of the element of L1 and L2 interleaved in order until one of them becomes nil, and then append that non-nil list at the end.

```
?- compose([a,b,c],[x,y,z],L).
L = [a,x,b,y,c,z]
?- compose([a,b,c],[x,y],L).
L = [a,x,b,y,c]
?- compose(L1,L2,[a,b,c]).
L1 = [] L2 = [a,b,c] ;
L1 = [a] L2 = [b,c] ;
L1 = [a,c] L2 = [b];
L1 = [a,b,c] L2 = []
```

8. **[15]** Define the predicate palindrome(Base, Result). You may assume that Result is always an even length list. However, you must show that your answer is the only possible answer.

```
?- palindrome([m,a,d],R).
R = [m,a,d,d,a,m] ;
no
?- palindrome(B,[n,i,s,s,i,n]).
B = [n,i,s] ;
no
```

Test your program with the following test cases, i.e, if you are using swipl, you can copy them into a test.in and then do “swipl < test.in”.

```
["main.pl"]. % ignore this if you are using web-app prolog
my_reverse([a,b,c,d], R).

my_reverse(O, [a,b,c,d]).

my_length([[a,b],[c,d],[e,f]], Y).
my_subset(atom, [a,[b],c], Y).

my_member(a,[a,b,c]).

my_member(Y,[a,b,c]).

my_intersect([a,b,c],[c,b], R).
my_intersect([a,b,c],[[c],d,b,e,a], R).
compute_change(33,Q,D,N,P).

compute_change(182,Q,D,N,P).
compose([a,b,c],[x,y,z],L).
compose([a,b,c],[x,y],L).
compose(L1,L2,[a,b,c]).
palindrome([m,a,d],R).
palindrome(B,[n,i,s,s,i,n]).
```

Submit:

- **submit the file “main.pl” that contains all your function definitions inside on Gradescope**