Reading Assignment. Read the following sections in the textbook:

- Sections 12.4–12.6: LP Computation Model and Extensions

Programming Homework. The goal of this assignment is to try out logic-programming in Prolog. The assignment is broken into three parts. Part 1 focuses on writing some basic queries and rules. Part 2 is to implement HW-7 (basic OCaml functions) using Prolog. Part 3 is to implement some basic recursive list processing functions (from HW-8) in Prolog. Each part will be implemented in a diferent file (`hw10a.pl`, `hw10b.pl`, and `hw11c.pl`).

Part 1: Queries. Use the `movies.pl` prolog file provided in the starter code for this assignment. The file represents a database (i.e., set of facts) containing information on movies, actors/actresses, and directors:

- `movies(M,Y)` states that a movie `M` was released in year `Y`.
- `director(M,D)` states that a movie `M` was directed by a director `D`.
- `actor(M,A,R)` states that actor `A` played the role `R` in the movie `M`.
- `actress(M,A,R)` states that actress `A` played the role `R` in the movie `M`.

Step 1. Using the `swipl` interpreter, load the `movies.pl` file and write the following queries. As an example, the following query finds the names of movies in the `movies.pl` database:

```
?- consult(`movies.pl').
% movies.pl compiled 0.03 sec, 2,597 clauses
true.

?- movie(M,_).
M = barton_fink ;
M = the_big_lebowski ;
M = blade_runner ;
... etc ...
```

1. Write a query to find movies that were released in 2006.
2. Write a query to find movies that were released in the 1980's (Hint: `Y < 1990` restricts `Y` values to be less than 1990.)
3. Write a query to find directors of movies released in 1998.

4. Write a query to find actors in a movie that they also directed.
5. Write a query to find movies in which Frances McDormand and Holly Hunter were co-stars.

Step 2. In the file `hw10a.pl`, implement the following rules. You should import (consult) `movies.pl` in `hw10a.pl` as follows:

```
:- consult(`movies.pl').
```

And then within the interpreter, you can load your file (which will also load `movies.pl`) as follows:

```
?- consult(`hw10a.pl').
```

6. Write rules for a relation `star(M,A)` such that `A` is either an actor or actress in movie `M`. (Note: Do <u>not</u> use the `` `;' `` operator.)
7. Write a rule for a relation `co_star(A1,A2)` such that `A1` and `A2` were two different stars in the same move. (Hint: to ensure two values are not equivalent use the `\==` comparison operator.)
8. Write a rule for a relation `starred_2(A,Y)` to find actresses and actors that starred in 2 different movies in the same year.

For questions 1–5, write your queries as comments in `hw10a.pl`. Be sure to fill in the file header comments and add comments stating the question and any additional information needed. For example:

```
% Question 1: Query to find movies released in 2006
% ...

% Question 2: ...
...

... and so on ...
```

Part 2: Basic Functions. For this part, you must implement and provide unit tests in Prolog for each of the 10 functions specified in HW-7 (see HW-7 for details of each function). Each function and your unit tests should go in `hw10b.pl`. To run your tests from the comman line, use:

```
$ swipl -g run_tests -t halt hw10b.pl
```

You can also run `swipl`, consult `hw10b.pl`, and then type `run_tests` from the query prompt. Additional details will be provided in Piazza. The following lists the structures that should be used for the corresponding HW-7 function.

1. `my_min(X,Y,M)` where `M` is the minimum of `X` and `Y`.
2. `my_median(X,Y,Z,M)` where `M` is the median of the other three arguments.
3. `my_triangle_area(Base,Height,Area)`
4. `my_circle_area(Radius,Area)`
5. `my_midpoint((X1,Y1), (X2,Y2), (X3,Y3))` where `(X1,Y1)` is the first coordinate, `(X2,Y2)` is the second coordinate, and `(X3,Y3)` is the midpoint between the two.
6. `my_manhattan_distance((X1,Y1), (X2,Y2), D)` where `(X1,Y1)` is the first point, `(X2,Y2)` is the second point, and `D` is the manhattan distance between them.
7. `my_euclidean_distance((X1,Y1), (X2,Y2), D)` where `(X1,Y1)` is the first point, `(X2,Y2)` is the second point, and `D` is the euclidean distance between them.
8. `my_range_sum(Start,End,Sum)` where `Start` is the starting value, `End` is the ending value, and `Sum` is the sum of the range.
9. `my_gcd(X,Y,D)` where `X` and `Y` are integer values and `D` is the greatest common divisor.
10. `my_even(X)` and `my_odd(X)`. Note that unlike in OCaml, these rules can be written separately (actually, they must be).

Finally, note that unit tests are placed in a separate "block" within your source code (as specified in the starter code). For instance, the following is an example of how to define a `my_max` relation and then corresponding unit tests.

```
% "function" definitions

my_max(X,Y,X) :- X >= Y.
my_max(X,Y,Y) :- Y > X.

... rest of "function" definitions" ...
```

```
% unit tests

:- begin_tests(all_tests).

test(my_max_1) :- my_max(1,2,2), !.
test(my_max_2) :- my_max(2,1,2), !.
test(my_max_3) :- my_max(1,1,1), !.

... rest of unit tests for "functions" ...

:- end_tests(all_tests).
```

Note above that the tests end with a "cut" (i.e., the "`!`" operator). A cut tells Prolog not to backtrack after it reaches the cut operator (i.e., we are "cutting" the rest of the branches from the proof tree). If we didn't include the cut, the testing framework would give a warning that more answers might exist. (Technically, we should use the cut in the rules themselves, but using the cut in the test is sufficient for this assignment). Be sure to provide enough test cases to ensure each relation is working correctly.

Part 3: List Functions. Similar to above, write the following Prolog versions of the list functions below from HW-8 in the file `hw10c.pl` along with corresponding unit tests. Note that each of these are recursively defined with multiple cases. Like with OCaml pattern matching, you must take care to clearly distinguish the cases from each other (especially since Prolog cases don't "shadow" each other—all cases are tried—and no warnings are given regarding shadowing).

1. The `my_last` function as the relation `my_last(Xs,X)` where `X` is the last element of the list `Xs`.
2. The `my_init` function as the relation `my_init(Xs,Ys)` where the list `Ys` contains all but the last element of the list `Xs`.
3. The `my_replace` function as the relation `my_replace((A,B),Xs,Ys)` where occurrences of `A` in the list `Xs` are being replace by `B` in `Ys`.
4. The `my_elem_sum` function as the relation `my_elem_sum(X,Ys,Sum)` where `X` is the element we want to add up in the list `Ys` to get the sum in `Sum`.
5. The `my_min` function as the relation `my_min(Xs,M)` where `M` is the minimum value in the list `Xs`.

Be sure to provide enough test cases to ensure each relation is working correctly.

Homework Submission. All homework must be submitted through GitHub Classroom. A link for each assignment will be posted on Piazza when the homework is assigned. Be sure all of your code is pushed by the due date (which you can double check for your repository using the GitHub website). Each programming assignment is worth **35 points**. The points are allocated based on the following.

- **Correct and Complete (25 points)**. Your code must correctly and completely do the requested tasks using the requested techniques. Note that for most assignments you will be provided a *partial* set of test cases to help you determine a *minimal level* of correctness. If your program fails any of the provided test cases you will only receive partial credit. *Note that passing the given test cases does not mean your work is complete nor correct.* Your assignment will also be graded with additional test cases (not provided to you) that will help the graders determine the extent of your solution and your final score. Note that for C++ code, correctness also implies properly handling the creation and deletion of dynamic memory (i.e., the absence of memory leaks).

- **Evidence and Quality of Testing (5 points)**. As part of your homework assignments you must develop additional test cases beyond those given to you to ensure your program is correct and complete. These test cases must be turned in with your assignment. You will be graded on the scope and quality of the additional test cases you provide.

- **Formatting and Comments (5 points)**. Your code must be formatted consistently and appropriately for the language used. For C++, you must follow the provided style guide (see the course webpage).You must also comment your code and test cases, which at a minimum must include a file heading (see examples provided), function comments, and meaningfully selected variable, class, and function names. See the assignment for style guides in other languages.