[Due Date: June, 4, 2pm]

CS 381, Spring 2019 — Homework 6 (Prolog)

Please follow carefully ***all*** of the following steps:

1. Put all the definitions into **one** file named `hw5.pl` (!!! **VERY IMPORTANT:** ***use lowercase letters*** !!!). This file should contain the shown definitions of the `when`, `where`, and `enroll` predicates at the beginning. **IMPORTANT:** Do ***not*** include any extraneous entries for these predicates! If you want to test your predicate definitions with additional examples, please do this in a separate file. The file that you submit should have ***exactly*** the data shown in the exercise—no more, no less.
   It is probably a good idea to download the file `hw5.pl` and to enter your predicate definitions into it.
2. Submit ***only one*** solution per team (each team can have up to 5 members), through the COE TEACH web site. List the names of all team members as a comment in the file.
3. Hand in a ***printed*** copy of your solution (***one copy*** per team) before class on June, 4, or make otherwise sure that the TAs receive the copy before the deadline. Make sure that all lines are readable on the printout.

Late submissions will ***not*** be accepted. Do ***not*** send solutions by email.

Exercise 1. Database Application

Consider a database about classes, class times, classrooms, and student enrollment, given in the form of Prolog facts.

```
when(275,10).          where(275,owen102).          enroll(mary,275).
when(261,12).          where(261,dear118).          enroll(john,275).
when(381,11).          where(381,cov216).           enroll(mary,261).
when(398,12).          where(398,dear118).          enroll(john,381).
when(399,12).          where(399,cov216).           enroll(jim,399).
```

Define the following derived Prolog predicates by one or more rules. Note that the shown goals are just examples. You should define the predicates so that it is possible to formulate goals with variables or constants at any argument position. ***Hint***: The inequality of, say two variables `X` and `Y`, can be expressed using the subgoal `X \= Y`.

(a) Define a predicate `schedule/3` that gives for a student the classrooms and times of his or her taken classes, that is, if you evaluate the goal `schedule(mary,P,T)`, Prolog should give the following result.

```
?- schedule(mary,P,T).

P = owen102
T = 10 ;

P = dear118
T = 12 ;
```

As another application of the `schedule` predicate, consider the goal `schedule(S,cov216,T)` that shows all students that are in the classroom `cov216` together with the corresponding time.

```
?- schedule(S,cov216,T).

S = john
T = 11 ;

S = jim
T = 12 ;
```

(b) Define a predicate `usage/2` that gives for a classroom all the times it is used. For example, the goal `usage(cov216,T)` should yield the following result.

```
?- usage(cov216,T).

T = 11 ;

T = 12 ;
```

The goal `usage(X,11)` should list all classrooms that are used at `11`.

(c) Define a predicate `conflict/2` that can compute conflicts in the assignment of classes to classrooms. A conflict exists if two different classes are assigned to one classroom for the same time. The arguments of the `conflict` predicate are two class names. You can use the goal `conflict(275,X)` (or `conflict(X,275)`) to find out any classes that are in conflict with the class `275`.

```
?- conflict(275,X).
false.
```

The goal `conflict(X,Y)` determines all pairs of possible conflicts.

(d) Define a predicate `meet/2` that can determine pairs of students that can meet in a classroom by either attending the same class or by having classes that are back to back in one classroom. The last condition means that a student Jim can meet any student who has a class that is in the same classroom and immediately follows Jim's class. (Note that your definition of `meet` doesn't have to be symmetric, that is, if students A and B can meet, then your implementation has to return `Yes` for `meet(A,B)` or `meet(B,A)`, but not necessarily for both calls. You can ignore the case when students are enrolled in conflicting classes.)

Exercise 2. List Predicates and Arithmetic

Note: Do ***not*** use the predefined predicates `flatten` and `nth`. You are allowed to use predefined predicates, such as `append` or `member`.

(a) Define a Prolog predicate `rdup(L,M)` to remove duplicates from an ordered list `L`. The resulting list should be bound to `M`. Note that `M` must contain each element of `L` exactly once and in the same order as in `L`. You can assume that `L` is an ordered list.

(b) Define a Prolog predicate `flat(L,F)` that binds to `F` the flat list of all elements in `L` (where `L` can be a possibly nested list). For example, `flat([a,b,[c,d],[],[[[e]]],f],L)` yields `L = [a,b,c,d,e,f]`.

(c) Define a Prolog predicate `project/3` that selects elements from a list by their position and collects them in a result list. For example, the goal `project([2,4,5],[a,b,c,d],L)` should produce the answer `L=[b,d]`. You can assume that the numbers in the first list are strictly inreasing, that is, your implementation does not have to care about situations like `project([1,1,2],...)` or `project([2,4,3],...)`.