```prolog
/*Michael Amann
 CSE 240 Richard Whitehouse
 TTh @ 3:30-4:15
 Prolog Lab2
 Made with XEmacs!*/

/*Given the following relation templates for a coffee club factbase:
manager(Name), true if Name is a manager.*/

/*Each person in this factbase in a manager*/
      manager(galron).
      manager(bellana).
      manager(worf).
      manager(harry).

/*bill(Name, Number, Amount), true if Name has been sent a bill
numbered Number for Amount. paid(Number, Amount, Date), true if a
payment of Amount was made on Date for the bill
numbered Number for the amount.*/

/*Each person in the "bill" fact base has received a bill*/

      bill(galron,1,50).
      bill(bellana,2,75).
      bill(worf,3,105).
      bill(harry,4,8).
      bill(galron,5,100).
      bill(harry,6,200).
      bill(ensign,7,120).

/*Each person in the "paid" factbase paid their bill on the stated date
for the state amount*/

      paid(1,50,031601).
      paid(2,75,032101).
      paid(3,100,030201).

/*Define views of the factbase (rules and queries) to answer the
following questions: Which manager has been sent a bill for less than
ten dollars?*/

/*Less than ten returns the name of the manager that received a bill
for an amount less than ten dollars. First the person is verified to be
a manager, then the database searches for a bill in the amount of less
than ten dollars that is associated with the name of that manager.*/

less_than_ten(Name) :- manager(Name),
                   bill(Name,Number,Amount),
                        Amount @< 10.

/*Who has been sent more than one bill?
The relation "billed" searches the database for persons who received a
bill. It then looks again for another bill that has the same name as
the first. After comparing whether the number on the two bills matches,
if it doesn't, that person's name is returned.*/
```

```
billed(Name):- bill(Name,Number,Amount),
                  bill(Name1,Number1,Amount1),
                        Name == Name1,Number \== Number1.

/*Who has made a payment that is less than the amount of their bill?
The relation "past_due" checks the two facts bases, bill and paid. When
the number in bill matches the number in paid, the amounts are
compared. If the Amount in bill is larger than the Amount in paid, the
name is returned.*/

      past_due(Name):- bill(Name,Number,Amount),
                  paid(Number,Amount2,Date),
                        Number == Number,
                              Amount @> Amount2.

/*Who has received a bill and either not paid it at all, or not paid
their bill prior to a specified date? In the relation "not paid," the
two fact bases bill and paid are traversed. If there is a bill for a
person an no entry whatsoever in the paid database, the name is
returned.*/

      not_paid(Name):- bill(Name,Number,Amount),
                  \+paid(Number,Amount,Date).

/*In the relation "paid_late," the two factbases are searched. When the
Number in bill matches the Number in paid, the date is compared to the
one supplied by the user. If the date is less than or equal the date
supplied, the name is returned.*/

      paid_late(Name,Y):- bill(Name,Number,Amount),
                  paid(Number,Amount,Date),
                        Date @=< Y.

/*The relation "dead_beats," combines "paid_late" and "not_paid" to
search for entries in the database in which a person has not paid their
bill at all or has not paid by a specified date.*/

      dead_beats(Name,Y):- not_paid(Name).
                    :- paid_late(Name,Y).

/*Define rules for the following relations:
Write your own times(N1, N2, Prod) which is true if Prod is the product
of N1 and N2.In the "times" relation, the number provided by the user
are evaluated to see if they are products of each other. If so, yes is
returned. */

      times(N1,N2,Prod):- Prod is N1* N2.

      /*Sample call | ?- times(3,2,6).
                  yes */

/*Write your own prefix(Prefix, List) rule which is true if Prefix is a
prefix of List. In the "prefix" relation, append is used to return the
first element of a list and compare it to the element supplied by the
user to determine it is a prefix of a list.*/
```

```
      prefix(X,Y):- append([Y],_,X).

      /*Sample call: | ?- prefix([a,b,c], a).
                  yes */

/*Write your own suffix(Suffix, List) rule which is true if Suffix is a
suffix of List. In the "suffix" relation, append is used to return the
last element of a list and compare it to the element supplied by the
user to determine if it is a suffix of a list.*/

      suffix(X,Y):- append(_,[Y],X).

      /*Sample call: | ?- suffix([a,b,c], c).
                  yes */

/*Write your own sublist(Sub, List) rule which is true if Sub is
sublist of List. In the "sublist" relation member is used to parse
through the two supplied list and assign the first of each list to a
value Z. If Z in List1 ever matches Z in List2, yes is returned.*/

      member(X,[X|_]).
      member(X,[_|Tail]):- member(X,Tail).
      sub_list(X,Y):- member(Z,X),
                        member(Z,Y),
                              Z==Z.

      /*Sample calls? | ?- sub_list([a,b,c],[b]).

                  yes

                  | ?- sub_list([a,b,c],[d]).
                  no */

/*Write your own append(List1, List2, List3) rule which is true if
List3 is formed by joining List1 and List2 (List1List2).*/

      addhead(List,Element,[Element|List]).
            my_append(X,Y,N):- member(Z,X),
                        member(A,X), addhead(Y,Z,N),addhead(Y,A,N).

      /*Sample call ?- my_append([a],[b,c,d],N).
                  N = [a,b,c,d]     */

/*Using only the append relation, formulate rules to:
Determine the third element of a list. In this example we are not
concerned with the first two elements of the list. We simply append the
third element of the list with nothing and assign it to X.*/

      third(X,Y):- append([_,_,Y],_,X).

/*Determine the last element of a list. In this example, the last
element of the rest of the list is assigned to X and returned.*/

      last(X,Y):- append(_,[Y],X).
```