```
/*Michael Amann
  CSE 240 TTh@ 3:14-4:30
  Richard Whitehouse
  Prolog Lab3 */


/*define the following relations on lists:

   1.delete(A, X, B) true if list B is the result of deleting a single
occurence of X from
   list A, ie: delete([3, 1, 4, 2], 4, [3, 1, 2]).

   Delete takes three parameters, a list, the element to be excluded and
the resulting list
   or an unbound variable. Passing the list to the append function, it
is broken in two
   excluding the element supplied that is to be removed. Somelist is
given the head of the
   list, and A is given the rest. Then Somelist and the rest of the list
are appended to
   form a new list less the given element.*/

      delete(A, X, B):-
            append(Somelist,[X|Xs], A),
                  append(Somelist, Xs, B).


/* 2.sorts(A, B)is true if list B is an ordered permutation of list A.
      Sample call
      ?- sorts([a,b,c,d],[b,d,c,a]).
      yes */

/* Sorts takes two parameters, either two lists or a list an an unbound
variable. The base
   case is two empty lists. Sorts first breaks up the first list into
Head and Tail (or X,Xs).
   Sorts then passes the rest of the first list and an unbound variable
Ys back to the relation.
   X is then deleted from Ys1 and the remaining list is placed in the
empty list Ys and returned
   */

      sorts([], []).
      sorts([X|Xs], Ys1):-
      sorts(Xs, Ys),
            select(X, Ys1, Ys).

/*Select(X,Xs,Ys) is true if Ys is the result of removing the first
occurence of X from Xs.
  Select accomplishes this task by first taking the supplied element,X,
and comparing it
  with the head of the supplied list. If X and the head match, the rest
of the list is passed
  to the unbound variable and the function returns. If X and the head do
not match, the list
  is passed as a formal parameter and broken up with the head being
saved to Y. Y is then
  placed at the head of the unbound variable and the rest of Y, Ys is
passed back to the
  function along with X and the rest of the unbound variable. When X and
```

```
the head finally do
  match, the rest of the list is saved to Xs, and the recursive calls
build the list back up
  excluding the selected element.*/

            select(X,[X|Xs],Xs).
                  select(X, [Y|Ys],[Y|Zs]):-
                  select(X, Ys, Zs).


/* 3.substitute(X, Y, L1, L2) where L2 is the result of substituting Y
for all occurrences of X
    in L1:
     substitute(a, x, [a,b,a,c], [x,b,x,c]) is true
     substitute(a, x, [a,b,a,c], [a,b,x,c]) is false */

 /* substitute(X,Y,Xs,Ys) is true if the list Ys is the result of
substituting Y for all
    occurrences of X in the list Xs.
    Substitute completes its task by continually comparing the heads of
two list to see if
    they meet the requirements of the relation. To begin, substitute
breaks up two lists and
    assings the heads to X and Y. The rest of the two lists are passed
back into the function.
    The heads of those list are then stored in Z and compared to X. If X
matches Z then the
    substitution is not complete and the relation fails. If there is no
match, the list is
    parsed through until the base case of the empty list is found, and
upon returning, the list
    is rebuilt and returned.*/

      substitute(X,Y,[],[]).
      substitute(X,Y,[X|Xs],[Y|Ys]):- substitute(X,Y,Xs,Ys).
      substitute(X,Y,[Z|Xs],[Z|Ys]):- X \==Z, substitute(X,Y,Xs,Ys).

/*   4.     no_doubles(L1, L2) where L2 is the result of removing all
duplicate elements from L1.
      ie: no_doubles([a,b,c,b], [a,c,b]) is true
      No_doubles accomplishes its task by breaking up the first list to
head and tail. The
      head is assigned to X and the tail to Xs, Ys is an unbound
variable. It then checks
      to see if the head element is a member of the rest of the list. If
it is, in which
      case, its a double, the rest of the list is sent back to the
relation an nothing is
      added to the unbound variable. The relation repeats, when the head
is not a member
      of the rest of the list, the head is double-checked to be a "non-
member" and that
      element is assigned to the head of the list Ys. This creates a new
list of
      "no_doubles. The relation ends with the base case of two empty
lists.*/

      no_doubles([], []).
      no_doubles([X|Xs], Ys):-
            member(X, Xs),
            no_doubles(Xs, Ys).
                  no_doubles([X|Xs], [X|Ys]):-
                  \+ member(X, Xs),
                        no_doubles(Xs, Ys).

      /*Member (X,Xs) is true if X is a member of the list Xs.*/
      member(X,[X|Xs]).
      member(X,[Y|Ys]):- member(X,Ys).


/*   5.Binary trees are represented by the functor tree
(Element,Left,Right), where Element is
     the element at the node, Left
     and Right are the left and right subtrees. The empty tree is
represented by the atom void.
     Using this notion, the tree:

                         a
                        / \
                       b   c

     is represented as:
     tree(a, tree(b, void, void), tree(c, void, void).
     Write a prolog program binary_tree(Tree) which is true if Tree is a
binary tree. */

/*
      binary_tree(Tree) :- Tree is a binary tree. This relation works by
checking each
      side of the tree to see if it has a left and a right child. The
relation stops when
      the value is void and no more children exitst.

      Sample call
      binary_tree(tree(a,tree(b,void,void),tree(c,void,void))).
      yes */

      binary_tree(void).
      binary_tree(tree(Element,Left,Right)) :-
            binary_tree(Left), binary_tree(Right).
```