Introduction to Functional Programming
Lent 2005
Suggested Exercises 1

1. Define a matrix to be a list of lists of real numbers all of equal length. Write ML functions to implement addition and multiplication of matrices.

For a challenge, use functionals (`map`, `foldl`, `foldr`, `zip`, etc.) and no explicit recursion. *Hint: you may find it helpful to implement a matrix transpose function.*

2. Use the following function to code an alternative version of merge-sort that doesn't use the `take` and `drop` functions. How does it compare in speed with the version given in the lecture? What accounts for the difference?

```
fun alts ([], l1, l2)       = (l1, l2)
  | alts ([x], l1, l2)      = (x::l1, l2)
  | alts (x::y::t, l1, l2) = alts (t, x::l1, y::l2)
```

(Based on Exercise 3.41 from *ML for the Working Programmer*)

3. Write definitions in ML for the following functions:

```
nodups :  ''a list -> ''a list
```

`nodups l` is the list containing all the elements of `l`, but without repetitions. So `nodups [1,2,3,2] = [1,2,3]`.

```
union :  ''a list * ''a list -> ''a list
```

which forms the union of two lists, i.e. the list containing all the elements of the two input lists, again without repetitions.

Why were equality type variables used in the above specifications? Can you define similar functions for arbitrary types?