SOLUTION NOTES

Foundations of Functional Programming 2003 Paper 5 Question 10 (ACN)

```
An empty list has no elements, so
   fun null f x = x

Here is a typical list called l written in ML-like syntax
   fun l f x = f a1 (f a2 (f a3 x));

Now a test for empty is easy!
   fun isempty l = l (\a. \b. false) true
and if l is empty this is obviously true, while if i is non-empty
then the \a.\b.false gets applied to 2 args so returns false.

   fun cons a0 l = \f.\x. f a0 (l f x)    [easy]

Extracting the head & tail of a non-empty list seems messier? Well
   fun head l = l (\a.\b.a) bottom

Note that head null -> bottom here (bottom is anything that diverges).

This just leaves tail.
Well here is a really grungy way of showing that it is possible!
  fun pair a b c = c a b
  fun left p = p K
  fun right p = p (K I)
  fun altnil = pair true ?
  fun altcons a b = pair false (pair a b)
  fun altcdr p = right (right p)
  fun altnull p = left p
What I have just done is to build a quite separate representation of
list structures!!!!!
  fun toAlt l = l altcons altnil
  recfun fromAlt l = if altnull l then null
     else cons (altcar l) (fromAlt (altcdr l)
Arrange to be able to convert reps.
  fun cdr x = fromAlt (altcdr (toAlt x))

recfun uses the fixed point operator Y. My solution here converts
```

```
from the NICE seeming functional representation of lists into a very
concrete one where CDR is easy, and it then converts back again. Oh
how horrid. Can anybody give me a much neater solution, please?



At the end map is going to be easy again

   fun map f l = \g.\x
      l (\a.\b. g (f a) b) x
```