2 Foundations of Computer Science (LCP)

The function perms returns all $n!$ permutations of a given n-element list.

```
fun cons x y = x::y;

fun perms [] = [[]]
  | perms xs =
      let fun perms1 ([],ys) = []
            | perms1 (x::xs,ys) =
                map (cons x) (perms (rev ys @ xs)) @
                perms1 (xs,x::ys)
      in  perms1 (xs,[])  end;
```

(*a*) Explain the ideas behind this code, including the function `perms1` and the expression `map (cons x)`. What value is returned by `perms [1,2,3]`? [7 marks]

(*b*) A student modifies `perms` to use an ML type of lazy lists, where `appendq` and `mapq` are lazy list analogues of `@` and `map`.

```
fun lperms [] = Cons ([], fn() => Nil)
  | lperms xs =
      let fun perms1 ([],ys) = Nil
            | perms1 (x::xs,ys) =
          appendq (mapq (cons x) (lperms (rev ys @ xs)),
                   perms1 (xs,x::ys))
      in  perms1 (xs,[])  end;
```

Unfortunately, `lperms` computes all $n!$ permutations as soon as it is called. Describe how lazy lists are implemented in ML and explain why laziness is not achieved here. [5 marks]

(*c*) Modify the function `lperms`, without changing its type, so that it computes permutations upon demand rather than all at once. [8 marks]

All ML code must be explained clearly and should be free of needless complexity.