22. Use recursion to define functions \( \text{orList} :: [\text{Bool}] \to \text{Bool} \) and \( \text{andList} :: [\text{Bool}] \to \text{Bool} \), where \( \text{orList} \) is true when at least one of the entries in the list is true, and \( \text{andList} \) is true when all of them are true.

23. Use recursion to define the \text{concat} function in terms of the ++ function. (If you want to test it on the interpreter, you will need to give it a different name from \text{concat}, since there is already a built-in \text{concat} function.)

24. Write a function \( \text{remove} :: \text{Eq a} \Rightarrow [a] \to [a] \to [a] \) such that \( \text{remove} \) gives returns a list of all elements of the list \( \text{ys} \) that are not elements of \( \text{xs} \). Write (at least) two versions: (1) using recursion, not using the elem function, and (2) using list comprehension involving the elem function (this enables you to give a one-line definition).

25. A list \( \text{list1} \) is called a sublist of another list \( \text{list2} \) if the elements of \( \text{list1} \) occur in order— but not necessarily as a block— in \( \text{list2} \). For example, "Ache." is a sublist of "A character string." Write a recursive function \( \text{subList} :: \text{Eq a} \Rightarrow [a] \to [a] \to \text{Bool} \) that tests whether a first list is a sublist of a second list.

26. Recall that a partition of \( n \) elements is a sum \( n = p_1 + p_2 + \cdots + p_m \) with \( p_1 \geq p_2 \geq \cdots \geq p_m \geq 1 \). For example, the seven partitions of 5 are 5, 4 + 1, 3 + 2, 3 + 1 + 1, 2 + 2 + 1, 2 + 1 + 1 + 1, and 1 + 1 + 1 + 1 + 1. Define \( P(n) \) to be the number of partitions of \( n \).
   (a) Define \( P(n, k) \) to be the number of partitions of \( n \) for which \( k \geq p_1 \), in particular \( P(n) = P(n, n) = 1 + P(n, n - 1) \). Prove that if \( k \leq n - 1 \) then \( P(n, k) = P(n, k - 1) + P(n - k, k) \).
   (b) Write a Haskell function \( \text{p} :: \text{Integer} \to \text{Integer} \to \text{Integer} \) so that \( \text{p n k} \) equals \( P(n, k) \).
   (c) Use (b) to write a Haskell function \( \text{partitions} :: \text{Integer} \to \text{Integer} \) so that \( \text{partitions n} \) equals \( P(n) \).

27. Download the Haskell script “binomial.hs” from the course website and test it. If you find any errors, correct them.