Functional programming

Exercises III

The countdown problem

1. Redefine the combinatorial function *choices* using a list comprehension rather than using composition, *concat* and *map*.
2. Define a recursive function isChoise :: Eq a => [a] -> [a] -> Bool that decides if one list is chosen from another, without using the combinatorial functions *perms* and *subs*. Hint: start by defining a function that removes the first occurrence of a value from a list.
3. What effect would generalizing the function *split* to also return pairs containing the empty list have on the behaviour of *solutions*?
4. Using the functions *choices*, *expr,* and *eval*, verify that there are 33665406 possible expressions over the numbers 1, 3, 7, 10, 25, 50 and that only 4672540 of these expressions evaluate successfully.
5. Similarly, verify that the number of expressions that evaluate successfully increases to 10839369 if the numeric domain is generalised to arbitrary integers. What do you have to modify in the program to do this?
6. Modify the final program to allow the use of exponentiation in expressions

Interactive programming

1. Redefine the function putStr :: String -> IO () using a list comprehension and the library function *sequence_ :: [IO a] -> IO ()*
2. Using recursion, define a version of putBoard :: Board -> IO () that displays nim boards of any size, rather than being specific to boards with just five rows of stars. Hint: first define an auxiliary function that takes the current row number as an additional argument.
3. In a similar manner to the first exercise, redefine the generalized version of *putBoard* using a list comprehension and *sequence_*.
4. Define an action adder :: IO () that reads a given number of integers from the keyboard, one per line, and displays their sum. For example:

   ```
   > adder
   how many numbers? 5
   1
   3
   5
   7
   9
   The total is 25
   ```

   Hint: start by defining an auxiliary function that takes the current total and how many numbers remain to be read as arguments. You will also likely need to use the library functions *read* and *show*.
5. Redefine the function *adder* using the function *sequence :: [IO a] -> IO [a]* that performs a list of actions and returns a list of the resulting values.
6. Using *getCharHidden*, define an action *readLine:: IO String* that behaves in the same way as getLine, except that it also permist the delete key to be used to remove characters. Hint: the delete character is `\DEL`, and the control character for moving the cursor back one space is `\b`.