Written problems

In each of these, simplify the expressions step-by-step to get the final value. If there is an error, say exactly what the problem is. If the function gives an infinite loop, explain in general terms what the output will be. You should show enough detail to make it clear that you know what is going on. In all cases, you should be able to check your answer by typing the expressions into GHCi.

1. \(\text{map} \ (+5) \ (\text{filter} \ \text{even} \ \{1..4\})\)

2. \(\text{filter} \ \text{even} \ (\text{map} \ (+5) \ \{1..4\})\)

3. \(\text{map} \ (\text{take} \ 4) \ \{\text{"haskell"}, \text{"curry"}, \text{"is"}, \text{"my"}, \text{"father"}\}\)

4. \(\text{filter} \ (\equiv \text{True}) \ (\text{map} \ (\text{even} \ . \ \text{length}) \ \{\text{"haskell"}, \text{"curry"}, \text{"is"}, \text{"my"}, \text{"father"}\})\)

5. \((\text{head} \ (\text{zipWith} \ .) \ \{\text{succ}, \ (\ast 2)\} \ [(\ast 3), \ (+2), \ \text{pred}]\}) \ 7\)

Programming problems

For all these functions, use recursion or higher order functions. Don’t use list comprehensions.

- From Homework #4, rewrite the function \(\text{digit7}\) in points-free style, and rewrite \(\text{square7}\) using \(\text{filter}\) (no list comprehension).

- Write a polymorphic function called \(\text{excluder}\) which takes two lists, and returns the second list after deleting any elements which are also in the first list. So:
  \(\text{excluder} \ \{1, 2, 3\} \ {5, 2, 6, 7, 3, 1, 0}\) is \(\{5, 6, 7, 0\}\)

- Here is a function defined by a list comprehension- rewrite it using \(\text{map}\) and \(\text{filter}\).
  \(\text{compy} \ y = [x^2 - 4\ast x \mid x \leftarrow [1..y], \ \text{mod} x 4 == 1]\)

- Write a function called \(\text{pascalN}\) which takes a number \(n\) and returns the infinite list of numbers which appear in Pascal’s triangle exactly \(n\) times. For example:
  \(\text{pascalN} \ 3\) is \(\{6, 20, 70\}\) and then computes for a long time. (I’m not sure what the next number is, or if it is known at all.)
  (There is no good way to tell when to stop searching to build this list, so your function will always compute endlessly.)
  (Use \(\text{Int}\) for all your types, even though this may produce inaccurate results for large numbers.)

- Using functions from class on 10/6, make a function called \(\text{collatzN}\) which takes a number \(n\) and returns the smallest number whose Collatz sequence has length exactly \(n\).