# Version 1 – basic questions

1. Complete the following set of rules in order to compute the sum of all even elements of a list.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

addEven nil => 0

x % 2 == 0

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

addEven x;xs => …

x % 2 != 0

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

addEven x;xs => …

1. Complete the following set of rules in order to remove all zero-valued elements of a list.  
     
   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

removeZeros … => nil

…

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

removeZeros … => …

…

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

removeZeros … => …

1. Complete the following set of rules in order to support evaluation of arithmetic expressions (sum and product only).  
     
   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

eval (value i) => i

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

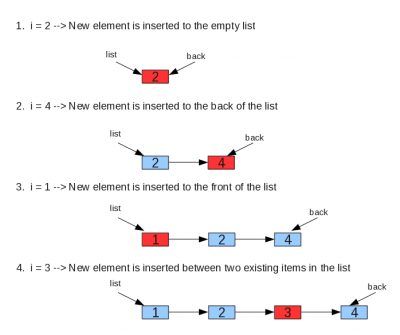
eval (a b) => …

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

eval (a b) => …

1. Complete the following set of rules in order to perform the insertion of an element in a sorted list, maintaining the ordering.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

insertSorted nil k => k;nil

x >= k

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

insertSorted x;xs k => k;x;xs

x < k

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

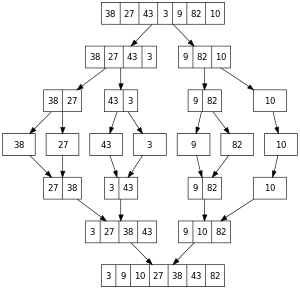
insertSorted x;xs k => …

# Version 2 – a more challenging problem

The *merge sort* algorithm works as follows (from wikipedia):

1. *Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted).*
2. *Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.*

An execution of merge-sort will look like the following:



In the picture above, the upper-half represents *split* steps, whereas the lower-half represents *merge* steps.

Define a system of rules which implements merge sort on lists.