CSE 341 | Section 5

Currying

a) Write a function `filter_by_example` that takes a function `f`, a value `x`, and a list `xs` in curried form. Upon applying the three arguments, the result of the function should be a new list that has all of the values from the original list that return the same result when `f` is applied to them as when `f` is applied to `x`.

b) Write a function `same_size_as` that takes a list and a list of lists in curried form and returns all of the lists in the second parameter that have the same size as the given list. Use `filter_by_example` in your answer.

c) Write a function `count_o` that takes a string and returns the number of occurrences of the lowercase letter `#"o"` in the given string. Our solution uses `List.filter` and `String.explode. List.foldl` is also possible.

d) Write a function `silly_application` that takes a list of strings and returns a new list of strings of all the strings in the given list that have the same number of occurrences of the letter o as "dogsarecool". Use `count_o` and `filter_by_example`.

e) Write a function `contains` that has type `''a -> ''a list -> bool` (notice the currying) and takes a first argument value, a second argument list, and returns true if the first argument is in the second argument. (Hint: use List.foldl)

f) Write a function `filter_unique` that takes a function, list of previous values, and an input list of values. If applying the given function to an input value results in a value not previously seen (not in the list of previous values), the input value should be added to the result list, and the result of applying the function should be added to the previous values list.

g) Write a function `unique_sums` that takes a list of lists of integers and returns a new list that contains lists that have unique summations. Use `filter_unique` in your answer.

h) Write a function `all_that_contain` that has type `''a -> ''a list list -> ''a list list` (notice the currying) which takes a value, and a list of lists, and returns a new list of all of the original lists that contain the given value.

i) Write a function `even_only` that takes a list of lists of ints and returns a new list of lists of ints that are the original lists with only even values. Use a val binding and some combination of `List.map` and `List.filter`

j) Write a function `even_only_not_empty` that returns the same thing as `even_only` except has no empty lists in its result. Our solution uses a fun binding, function composition, and calls to `List.filter` and `even_only`

Modules

(a) Below on the left are various lines of code that belong in the signature and module skeletons on the right. Your job is to discern which lines belong in the RATIONAL signature and which belong in the Rational module. For the sake of space, the full expression in the function bindings has been replaced with a comment of (* function_name body *)

Lines of Code	signature and module skeleton

a) `val make_frac : int * int -> rational`
 	
b) `fun toString (x,y) = (* to_string body *)`
 	
c) `type rational`
 	
d) `val toString : rational -> string`
 	
e) `fun Whole i = (i,1)`
 	
f) `exception BadFrac`
 	
g) `type rational = int * int`
 	
h) `fun add ((a,b),(c,d)) = (* add body *)`
 	
i) `val add : rational * rational -> rational`
 	
j) `val Whole : int -> rational`
 	
k) `fun make_frac (x,y) = (* make_frac body *)`
	`signature RATIONAL =`
	`sig`
	
	`end`
	
	`structure Rational :> RATIONAL =`
	`struct`
	
	`end`