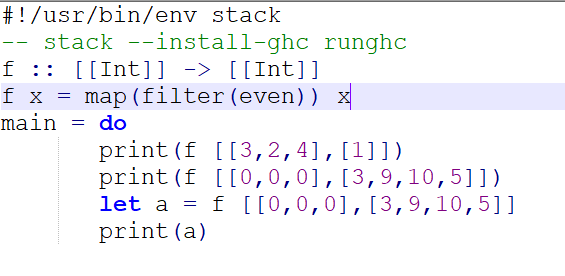
Lawrence Allen

Homework 2

2.1

a)

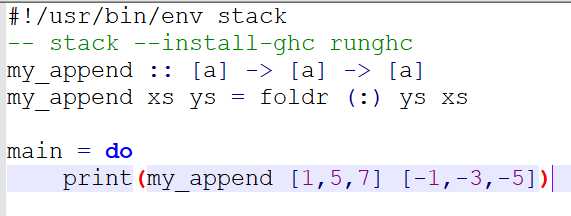
Code:



Output:



b)



Output:



2.2)

a) Defines a tree, which is either a leaf or a Node and two trees, which represent the connection between the node and its two children. A leaf or a node might be empty, but if it's not, they're all the same type. But that type can be anything.

b)

#!/usr/bin/env stack

-- stack --install-ghc runghc

data MTree a = MLeaf (Maybe a)

| MNode (Maybe a) (MTree a) (MTree a)

deriving (Show)

mTreeMap :: (a -> b) -> MTree a -> MTree b

mTreeMap f (MLeaf Nothing) = MLeaf Nothing

mTreeMap f (MLeaf (Just a)) = MLeaf (Just(f(a)))

mTreeMap f (MNode Nothing t1 t2) = MNode Nothing (mTreeMap f t1) (mTreeMap f t2)

mTreeMap f (MNode (Just n) t1 t2) = MNode (Just(f(n))) (mTreeMap f t1) (mTreeMap f t2)

mTreeFilter :: (a -> Bool) -> MTree a -> MTree a

mTreeFilter f (MLeaf Nothing) = MLeaf Nothing

mTreeFilter f (MLeaf (Just a)) = if f(a)

then MLeaf(Just(a))

else

MLeaf(Nothing)

mTreeFilter f (MNode Nothing t1 t2) = MNode Nothing (mTreeFilter f t1) (mTreeFilter f t2)

mTreeFilter f (MNode (Just a) t1 t2) = if f(a)

then (MNode (Just(a)) (mTreeFilter f t1) (mTreeFilter f t2))

else (MNode Nothing (mTreeFilter f t1) (mTreeFilter f t2))

main =

do

let a = MNode Nothing (MNode (Just 3) (MLeaf (Just 2)) (MLeaf (Just 1))) (MNode (Just 2) (MLeaf (Just 3)) (MLeaf (Just 4)))

print "a:"

print a

let b = mTreeMap (\* 2) a

print "b:"

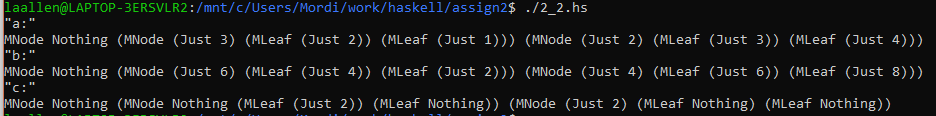
print (b)

print "c:"

let c = mTreeFilter(==2) a

print (c)

Output:



2.3)

Prove: map f (map g l) = map g (map f l)

Know: f(g(x)) = g(f(x)) for all numbers

Base Case: x = [], x is an empty list

By how map is defined, we know that:

map g x = []

and

map f x = []

so:

map f (map g x) = map f ([]) = []

and

map g (map f x) = map g ([]) = []

Thus:

map f (map g x) = map g (map f x)

Inductive hypothesis:

map f (map g n) = map g (map f n), where n is a list of arbitrary size

Prove:

map f (map g m) = map g (map f m), where m is a list of size(n) + 1

Proof:

map f (map g m) = map f (map g m:n), where m:n is the first element of m, followed by the other n elements

map f (map g m:n) = f(g(m)) ++ map f (map g n), by definition of map

f(g(m)) ++ map f (map g n) = g(f(m)) ++ map f (map g n), by the Know condition

g(f(m)) ++ map f (map g n) = g(f(m)) ++ map g (map f n), by Inductive hypothesis

g(f(m)) ++ map g (map f n) = map g (map f m:n), by definition of concatenation, and map

Thus:

map f (map g m) = map f (map g m)