COMP3007A(Fall 2018) – "Programming Paradigms"
Practice Question for Haskell Structural Induction

Question 1 :

For this question, prove that *nothingspecial* will provide the same result of the combination of different functions in the following :

```
nothingSpecial list =  myFoldr (myFilter list even) (+) 0
```

These are the function declaration that you might need to consider for your proof. Show every step in your proof and use the source code line labels whenever you use equational reasoning.

```
[nsb]    nothingSpecial [] =  0
                 nothingSpecial (h:t)
[nsr1]       | (even h) = (nothingSpecial t) + h
[nsr2]       | otherwise = (nothingSpecial t)

[frb]    myFoldr [] _ init = init
[frr]    myFoldr (h:t) oper init = oper h (myFoldr t oper init)

[ftb]    myFilter [] _ = []
                 myFilter (h:t) cond
[ftr1]       | (cond h) = h :: (myFilter t cond)
[ftr2]       | otherwise = (myFilter t cond)
```

COMP3007A(Fall 2018) – "Programming Paradigms"
Practice Question for Haskell Structural Induction

Question 2 :

For this question, prove that *myLast* can be rewritten by other functions like:

myHead (myReverse list) = myLast list

These are the function declaration that you might need to consider for your proof. Show every step in your proof and use the source code line labels whenever you use equational reasoning.

```
[reb]   myReverse [] = []
[rer]   myReverse (h:t) = (myReverse t) ++ [h]

[hdb]   myHead (h:_) = h

[ltb]   myLast [x] = x
[ltr]   myLast (h:t) = myLast t
```

COMP3007A(Fall 2018) – "Programming Paradigms"
Practice Question for Haskell Structural Induction

Question 3 :

For this question, prove that you can use other functions to achieve similar feature for *elementAt* like:

myTake 0 (myDrop n list) = [elementAt n list]

These are the function declaration that you might need to consider for your proof, *myTake* and *myDrop* might not be the same implementation of the built-in one, but the it will be valid for the proof. Show every step in your proof and use the source code line labels whenever you use equational reasoning.

```
[teb]   myTake 0 (h:_) = [h]
[ter]   myTake n (h:t) = h : (myTake (n - 1) t)

[dpb]   myDrop 0 list = list
[dpr]   myDrop n (h:t) = (myDrop (n - 1) t)

[eab]   elementAt 0 (h:_) = h
[ear]   elementAt n (h:t) = (elementAt (n - 1) t)
```

COMP3007A(Fall 2018) – "Programming Paradigms"
Practice Question for Haskell Structural Induction

Question 4 :

For this question, we are going to explore more inefficient way to produce *myLast*, it surely will not apply to the real life situation but it is fun for me to create this kind of question. Try to proof that:

myLast list = elementAt ((myLength list) – 1) list

These are the function declaration that you might need to consider for your proof, some of the functions you might see it in the previous questions already but I will just type again here. Show every step in your proof and use the source code line labels whenever you use equational reasoning.

```
[ltb]   myLast [x] = x
[ltr]   myLast (h:t) = myLast t

[lhb]   myLength [] = 0
[lhr]   myLength (h:t) = (myLength t) + 1

[eab]   elementAt 0 (h:_) = h
[ear]   elementAt n (h:t) = (elementAt (n - 1) t)
```

COMP3007A(Fall 2018) – "Programming Paradigms"
Practice Question for Haskell Structural Induction

Question 5 :

Most of the questions above rely on the basic data structure list in Haskell, this time, I would like to use my own data structure. This should be similar and probably easier than the assignment 4 one. Proof that:

count tree >= height tree

These are the function declaration that you might need to consider for your proof, this tree is not exactly the same from the assignment but the concept is similar. I think that this will not be super hard for you to understand. Show every step in your proof and use the source code line labels whenever you use equational reasoning.

```
datatype MeaninglessTree = NothingNode
        | (SomethingNode MeaninglessTree MeaninglessTree)

[ctb]       count NothingNode = 1
[ctr]       count (SomethingNode a b) = (count a) + (count b)

[htb]       height NothingNode = 1
[htr]       height (SomethingNode a b)
                              = (max (height a) (height b)) + 1

            max a b
[mx1]               | a > b = a
[mx2]               | otherwise = b
```