Name: Wen Chuan Lee
Student ID: 4927941 (leex7095)
Class: CSCI 2041
Title: Homework 8: Lazy Evaluations - Solutions

(let t be true and f be false in this question)

Question 2:

Evaluate `andl (t::f::t::t::[])`

Call by Value

```
andl (t::f::t::t::[])
= foldl and t (t::f::t::t::[])
= foldl and (and t t) (f::t::t::[])
= foldl and t (f::t::t::[])
= foldl and (and t f) (t::t::[])
= foldl and f (t::t::[])
= foldl and (and f t) (t::[])
= foldl and f (t::[])
= foldl and (and f t) []
= foldl and f []
= f           (false)
```

Call by Name

```
andl (t::f::t::t::[])
= foldl and t (t::f::t::t::[])
= foldl and (and t t) (f::t::t::[])
= foldl and (and (and t t) f) (t::t::[]))
= foldl and (and (and (and t t) f) t) (t::[])
= foldl and (and (and (and (and t t) f) t) t) []
= and (and (and (and t t) f) t) t
= and (and (and t f) t) t
= and (and f t) t
= and f t
= f           (false)
```

Evaluate `andr (t::f::t::t::[])`

Call by Value

```
andr (t::f::t::t::[])
= foldr and t (t::f::t::t::[])
= and t (foldr and t (t::f::t::t::[]))
= and t (and t (foldr and t (f::t::t::[])))
= and t (and t (and f (foldr and t (t::t::[]))))
= and t (and t (and f (and t (foldr and t (t::[])))))
= and t (and t (and f (and t (and t (foldr and t [] )))))
= and t (and t (and f (and t (and t t))))
= and t (and t (and f (and t t)))
= and t (and t (and f t)))
= and t (and t f)
= and t f
= f          (false)
```

Call by Name

```
andr (t::f::t::t::[])
= foldr and t (t::f::t::t::[])
= and t (foldr and t (t::f::t::t::[]))
= and t (and t (foldr and t (f::t::t::[])))
= and t (and t (and f (foldr and t (t::t::[]))))
= and t (and t f)
= and t f
= f          (false)
```

Explain which one is most efficient and why

Call by name semantics using andr is most efficient. This is because the expression is evaluated ‘outside in’ in which once the and function finds a false statement (not true), it will evaluate into the else-clause of the *and* function, then stop evaluating the rest of the list, saving computations. Call by name and call by value on *andl* does not make a difference as the entire expression still needs to be evaluated since *and* functions are nested inside the fold function and requires the rest of the expression to be evaluated.