Recitation Session 8, Solutions

Associativity.

Lemma 1: `flatMap` distributes over concatenation. For all `a: IList[A]`, `b: IList[A]` and `f: A => B`,
`(a ++ b).flatMap(f) == a.flatMap(f) ++ b.flatMap(f)`

By structural induction on `a: IList`.

Base case, `a = INil()`: `(INil() ++ b).flatMap(f) == b.flatMap(f) == INil() ++ b.flatMap(f) == (INil().flatMap(f) ++ b.flatMap(f)`

Induction case, `a = ICons(h, t)`

Suppose `(t ++ b).flatMap(f) == t.flatMap(f) ++ b.flatMap(f) (IH)`

```
   (ICons(h, t) ++ b).flatMap(f)
== (ICons(h, t) match {                        // Definition of ++
     case INil() => that
     case ICons(h, t) => ICons(h, t ++ b)
   }).flatMap(f)
== ICons(h, t ++ b).flatMap(f)                 // Simplification
== ICons(h, t ++ b) match {                    // Definition of flatMap
     case INil() => INil()
     case ICons(h', t') => f(h') ++ t'.flatMap(f)
   }
== f(h) ++ (t ++ b).flatMap(f)                 // Simplification
== f(h) ++ (t.flatMap(f) ++ b.flatMap(f))      // IH
== (f(h) ++ t.flatMap(f)) ++ b.flatMap(f)      // By associativity of concatenation
== (ICons(h, t) match {                        // Simplification
     case INil() => INil()
     case ICons(h, t) => f(h) ++ t.flatMap(f)
   }) ++ b.flatMap(f)
== ICons(h, t).flatMap(f) ++ b.flatMap(f)      // Definition of flatMap
```

This concludes the proof of Lemma 1.

Associativity is then showed By structural induction on `e: IList`.

Base case, `e = INil: INil.flatMap(f).flatMap(g) == INil == INil.flatMap(x => f(x).flatMap(g))`

Induction case, `e = ICons(h, t)`:

Suppose `t.flatMap(f).flatMap(g) == t.flatMap(x => f(x).flatMap(g)) (IH)`

```
   ICons(h, t).flatMap(f).flatMap(g)
== (ICons(h, t) match {                               // Def. of flatMap
     case INil() => INil()
     case ICons(h, t) => f(h) ++ t.flatMap(f)
   }).flatMap(g)
== (f(h) ++ t.flatMap(f)).flatMap(g)                  // Simplification
== f(h).flatMap(g) ++ t.flatMap(f).flatMap(g)         // By Lemma 1
== f(h).flatMap(g) ++ t.flatMap(x => f(x).flatMap(g))// IH
== ICons(h, t) match {                                // Simplification
     case INil() => INil()
     case ICons(h, t) => f(h).flatMap(g) ++ t.flatMap(x => f(x).flatMap(g))
   }
== ICons(h, t).flatMap(x => f(x).flatMap(g))          // Def. of flatMap
```

Left unit.
Lemma 2: for all `l: IList`, `l ++ INil() == l`
By structural induction on `l: IList`.
Base case, `e = INil()`: `INil() ++ INil() == INil()`
Induction case, `e = ICons(h, t)`:
Suppose `t ++ INil() == t (IH)`

```
== ICons(h, t) ++ INil()
== ICons(h, t) match {         // Definition of ++
     case INil() => that
     case ICons(h, t) => ICons(h, t ++ INil())
   }
== ICons(h, t ++ INil())       // Simplification
== ICons(h, t)                 // IH
```

This concludes the proof of Lemma 2.
Left unit is shown using a direct proof.

```
   unit(x).flatMap(f) == f(x)
== ICons(x, INil).flatMap(f) // Definition of unit
== ICons(x, INil) match {    // Definition of flatMap
     case INil() => INil()
     case ICons(h, t) => f(h) ++ t.flatMap(f)
   }
== f(x) ++ INil().flatMap(f) // Simplification
== f(x) ++ INil()            // Definition of flatMap
== f(x)                      // By Lemma 2
```

Right unit.
By structural induction on `e: IList`.
Base case, `e = INil()`: `INil().flatMap(unit) == INil()`
Induction case, `e = ICons(h, t)`:
Suppose `t.flatMap(unit) == t (IH)`

```
   ICons(h, t).flatMap(IList.singleton)
== ICons(h, t) match {               // Definition of flatMap
     case INil() => INil()
     case ICons(h, t) => unit(h) ++ t.flatMap(unit)
   }
== unit(h) ++ t.flatMap(unit)       // Simplification
== ICons(h, INil) ++ t.flatMap(unit)// Definition of unit
== ICons(h, INil) ++ t              // IH
== ICons(h, INil) match {           // Definition of ++
     case INil() => t
     case ICons(h', t') => ICons(h', t' ++ t)
   }
== ICons(h, INil() ++ t)            // Simplification
== ICons(h, t)                      // Definition of ++
```