COMP2600 Tutorial Solution
Structural Induction

Question 1.1

Base Case

```
[] ++ (ys ++ zs)          = ([] ++ ys) ++ zs

[] ++ (ys ++ zs)          = ys ++ zs                        -- by A1
                          = ([] ++ ys) ++ zs                -- by A1
```

Step Case

Assume

```
   as ++ (ys ++ zs)       = (as ++ ys) ++ zs               -- (IH)
```

Prove

```
  (a:as) ++ (ys ++ zs)    = ((a:as) ++ ys) ++ zs

  (a:as) ++ (ys ++ zs)    = a : (as ++ (ys ++ zs))         -- by A2
                          = a : ((as ++ ys) ++ zs)         -- by IH
                          = (a : (as ++ ys)) ++ zs         -- by A2
                          = ((a:as) ++ ys) ++ zs           -- by A2
```

Question 1.2

Base Case

```
  elem z ([] ++ ys)       = elem z [] || elem z ys

  elem z ([] ++ ys)       = elem z ys                      -- by A1
                          = False || elem z ys             -- by O2
                          = elem z [] || elem z ys         -- by E1
```

Step Case

Assume

```
  elem z (as ++ ys)       = elem z as || elem z ys         -- (IH)
```

Prove

```
  elem z ((a:as) ++ ys)   = elem z (a:as) || elem z ys
```

Case `z == a`

```
  elem z ((a:as) ++ ys)   = elem z (a : (as ++ ys))        -- by A2
                          = True                           -- by E2
                          = True || elem z ys              -- by O1
                          = elem z (a:as) || elem z ys     -- by E2
```

Case `z /= a`

```
    elem z ((a:as) ++ ys)   = elem z (a : (as ++ ys))      -- by A2
                            = elem z (as ++ ys)            -- by E3
                            = elem z as || elem z ys       -- by IH
                            = elem z (a:as) || elem z ys   -- by E3
```

Question 1.3

The 'hard structural induction' really is quite hard. It's worth preparing one's self for it. The main point is for them to actually see that even some simple theorems are hard to show mathematically. It would be rewarding if you had students who could solve it by hemselves, though.

What they should successfully do on their own is reduce the step case goal to

```
reverse (reverse as ++ [a]) = (a : as)
```

before getting quite stuck.

Either of the following two lemmas, which one can easily prove by induction, can revive this stuck proof.

- `reverse (ys ++ [a]) = a :  (reverse ys)`
- `reverse (as ++ ys) = reverse ys ++ reverse as`

As far as alternative definitions of reverse that might make the problem directly soluble, the other standard defn is shown here. It's still difficult, you use several lemmas. These are explored below under the material relating to `slinky`

```
reverse x = reverse2 x []
where reverse2 []     ys = ys
      reverse2 (x:xs) ys = reverse2 xs (x:ys)
```

Question 2

Base Case

```
    revT (revT Nil)         = Nil

    revT (revT Nil)         = revT Nil          -- by T1
                            = Nil               -- by T1
```

Step Case

Assume

```
    revT (revT a1)          = a1                              -- (IH1)
    revT (revT a2)          = a2                              -- (IH2)
```

Prove

```
revT (revT (Node x a1 a2)) = Node x a1 a2

revT (revT (Node x a1 a2)) = revT (Node x (revT a2) (revT a1))        -- by T2
                           = Node x (revT (revT a1)) (revT (revT a2)) -- by T2
                           = Node x a1 a2                         -- by IH1, IH2
```

Question 3.1

Do this one by induction on `xs`, need to use

`P(xs) = ∀ys. slinky (slinky xs ys) zs = slinky ys (xs ++ zs)`

<u>Base Case</u>

```
    slinky (slinky [] ys) zs      = slinky ys ([] ++ zs)

    slinky (slinky [] ys) zs      = slinky ys zs              -- by S1
                                  = slinky ys ([] ++ zs)      -- by A1
```

<u>Step Case</u>

Assume `∀ys. slinky (slinky as ys) zs = slinky ys (as ++ zs) -- (IH)`

Prove `∀ys. slinky (slinky (a:as) ys) zs = slinky ys ((a:as) ++ zs)`

```
  slinky (slinky (a:as) ys) zs  = slinky (slinky as (a:ys)) zs  -- by S2
                                = slinky (a:ys) (as ++ zs)      -- by IH (*)
                                = slinky ys (a:(as ++ zs))      -- by S2
                                = slinky ys ((a:as) ++ zs)      -- by A2
```

(*) Note, ys in the IH is instantiated to $a : ys$ when it is used in the proof

Question 3.2 Lemma (b)

Do this one by induction on `ys`, need to use

`P(ys) = ∀zs. slinky xs (slinky ys zs) = slinky (ys ++ xs) zs`

<u>Base Case</u>

```
    slinky xs (slinky [] zs)         = slinky ([] ++ xs) zs

    slinky xs (slinky [] zs)         = slinky xs zs            -- by S1
                                     = slinky ([] ++ xs) zs    -- by A1
```

<u>Step Case</u>

Assume `∀zs. slinky xs (slinky as zs) = slinky (as ++ xs) zs -- (IH)`

Prove `∀zs. slinky xs (slinky (a:as) zs) = slinky ((a:as) ++ xs) zs`

```
  slinky xs (slinky (a:as) zs) = slinky xs (slinky as (a:zs)) -- by S2
                               = slinky (as ++ xs) (a:zs)     -- by IH (*)
                               = slinky (a:(as ++ xs)) zs     -- by S2
                               = slinky ((a:as) ++ xs) zs     -- by A2
```

(*) Note, zs in the IH is instantiated to $a : zs$ when it is used in the proof

Question 3.2 Lemma (c)

Do this one by induction on `xs`, need to use

```
P(xs) = ∀ys.  slinky xs (ys ++ zs) = slinky xs ys ++ zs
```

<u>Base Case</u>

```
slinky [] (ys ++ zs)          = slinky [] ys ++ zs

slinky [] (ys ++ zs)          = ys ++ zs                    -- by S1
                              = slinky [] ys ++ zs          -- by S1
```

<u>Step Case</u>

Assume `∀ys.  slinky as (ys ++ zs) = slinky as ys ++ zs -- (IH)`

Prove `∀ys.  slinky (a:as) (ys ++ zs) = slinky (a:as) ys ++ zs`

```
slinky (a:as) (ys ++ zs)   = slinky as (a:(ys ++ zs))   -- by S2
                           = slinky as ((a:ys) ++ zs)   -- by A2
                           = slinky as (a:ys) ++ zs     -- by IH (*)
                           = slinky (a:as) ys ++ zs     -- by S2
```

(*) Note, ys in the IH is instantiated to $a : ys$ when it is used in the proof

Question 3.3

Take:

```
reverse xs  = slinky xs []                                  -- (Slinkify)
```

Rewrite:

```
reverse (reverse xs)          = slinky (slinky xs []) []   -- by Slinkify
                              = slinky [] (xs ++ [])       -- by 3.1a
                              = xs ++ []                   -- by S1
                              = xs                         -- easy lemma
```