Prove

```
(implies (and (endp x)
              (tlp x))
         (equal (app x nil)
                x))
```

```
(app x nil)
= { Def. app }
(if (endp x)
    nil
    (cons (car x)
          (app (cdr x) y)))
= { A1, if-true axiom }
nil
= { D1, see below }
x
```

Context

A1: `(endp x)`
A2: `(tlp x)`
The context can be expanded, using any rule of inference. Here we will use the def.axiom of tlp, if-true axiom and the above already known true statements(assumptions I started with) to deduce:
D1: `(equal x nil)` {*A2,def. Tlp,A1*}

Here is a simple technique, the idea is one can grow the context(which is a list of true statements we can use in the particular(current) proof(not anywhere else)) using our 3 Rules of Inferences.

Recall that Rules of inferences enable one to **deduce** new statements using existing true statements, but if we deduce something from temporarily true statements(assumptions) that are true only in the current context, then our deductions(e.g D1) also are temporarily true (i.e only in the current context/proof).

Let see if we can deduce D1 from the existing assumptions in my context, to get a larger context(the larger the context, the more easier my proof):

```
(tlp x)
=> { Def. tlp }
(if (endp x)
  (equal x nil)
  (tlp (cdr x)))
=> { A1, if-true axiom }
(equal x nil)
```

Since I used, A2, Def. Tlp and A1 to deduce D1, I will mention it in my context, see above, the context has been modified to include D1 and the reasons are also mentioned in curly brackets.

Now let's prove

```
(implies (and (consp x)
              (tlp x)
              (equal (app (cdr x) nil)
                     (cdr x)))
         (equal (app x nil)
                x))
```

Context
A1: `(consp x)`
A2: `(tlp x)`
A3: `(equal (app (cdr x) nil) (cdr x))`

```
(app x nil)
= { Def app, A1, def. endp, if-false }
(cons (car x)
      (app (cdr x) nil))
= { A3 }
(cons (car x)
      (cdr x))
= { (new)Cons axiom, A1, Prop. Deduction }
x
```

Here's the new **Cons axiom** I just used:

```
(implies (consp x)
         (equal (cons (car x)
                      (cdr x))
                x))
```

Given the following definitions:

```
(defun dup (x)
  (if (endp x)
    nil
    (cons (car x)
          (cons (car x)
                (dup (cdr x))))))
```

(dup '(a 2 3 d)) evaluates to (a a 2 2 3 3 d d)

```
(defun len (A)
  (if (endp A)
      0
     (+ 1 (len (cdr A)))))
```

We would like to prove some cases of the property that the length of the dup list is twice the length of the original list.

Prove the following:

```
(and (implies (endp x)
              (equal (* 2 (len x)) (len (dup x))))
     (implies (and (not (endp x))
                   (equal (* 2 (len (cdr x)))
                          (len (dup (cdr x)))))
              (equal (* 2 (len x)) (len (dup x)))))
```

Whenever we have to prove a conjunction, we just prove each subformula independently. Why? Simple, to show p /\ q true, we need to show both p **and** q true.

Subgoal 1 is to prove

```
(implies (endp x)
         (equal (* 2 (len x)) (len (dup x))))
```

Proof:
Since we are proving an implication, we simply assume everything in our antedecent as true, so our context is:

Context
1. `(endp x)`

```
(equal (* 2 (len x)) (len (dup x)))
<= {Def. Dup, A1, if-true}
(equal (* 2 (len x)) (len nil))
<= {def. Len, A1, if-true}
(equal (* 2 0) (len nil))
<= {Evaluation, arithmetic}
(equal 0 0)
<= {Equality axiom built into ACL2 logic}
t
```

We proved the first subgoal, now we will prove the second subgoal. As always we will build a new context, populate it with all the formulas in the antecedent of the implication, by the way, such formulas are called **assumptions**, thats why I name them A1, A2 and so on.
Note again, to prove (implies (and A1 A2 … An) C) all we need to do is prove C (the consequent) assuming A1, A2, ..,,An which all go into my context of the proof. The context of the proof is a placeholder where you keep all the “true statements” you can use in the current proof. So A1, A2, ...An are all true statements while proving C, but they are not true statements in general. Read the above 5 times. Therefore you **cannot** instantiate assumptions, because they are not theorems, they are only true in the current context/proof. Read the last line 10 times. Thank you for your patience. But could you read.....never mind.

Subgoal 2:

```
(implies (and (not (endp x))
              (equal (* 2 (len (cdr x)))
                     (len (dup (cdr x)))))
             (equal (* 2 (len x)) (len (dup x)))))
```

Proof:

Context

```
A1: (not (endp x))
A2: (equal (* 2 (len (cdr x)))
           (len (dup (cdr x)))))
```

Lets start with LHS and try to prove it equal to the RHS, this is slightly different from what I did in class, use whatever is more comfortable and easier to work with. Remember the direction in which your proof should proceed should be guided by your context and the lemmas(previously proved theorems) that you can use.

LHS:

```
 (* 2 (len x))
={ def. Len, A1, if-false }
 (* 2 (+ 1 (len (cdr x)))
={ Arithmetic }
 (+ 2 (* 2 (len (cdr x)))
={ A2, equals for equals }
 (+ 2 (len (dup (cdr x)))
={arithmetic}
 (+ 1 (+ 1 (len (dup (cdr x)))
={Use Instantiation of len with ((A (cons (car x) (dup (cdr x)))))}
 (+ 1 (len (cons (car x) (dup (cdr x)))))
Lets call (cons (car x) (dup (cdr x)) as B here
={Use Instantiation of len with ((A (cons (car x) B)))}
 (len (cons (car x) (cons (car x) (dup (cdr x)))))
={def. dup, A1}
 (len (dup x))
```

This might seem strange, but remember reverse distribution:
a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c) Distribution
(a ∨ b) ∧ (a ∨ c) = a ∨ (b ∧ c) Reverse Distribution

Actually I shouldnt have fancy name like reverse distribution, since its an equality and we know equality is symmetric I.e (equal a b) iff (equal b a), we should be comfortable with using both directions. So we can open up definitions but also close i.e the other direction. So in the step where

I reason "*Use Instantiation of len with ((A (cons (car x) (dup (cdr x)))))",* here is what is actually happening:

```
(equal (len A)
       (if (endp A)
           0
         (+ 1 (len (cdr A)))
```

Since this is an axiom, any instantiation is also a theorem, so if we apply the above subsitution we deduce:

```
(equal (len (cons (car x) (dup (cdr x))))
       (if (endp (cons (car x) (dup (cdr x))))
           0
         (+ 1 (len (cdr (cons (car x) (dup (cdr x))))))
```

But from consp-cons, if-false and cdr-cons axioms we deduce:

```
(equal (len (cons (car x) (dup (cdr x))))
       (+ 1 (len ((dup (cdr x))))
```

This by symmetry of equality is same as:

```
(equal (+ 1 (len ((dup (cdr x))))
       (len (cons (car x) (dup (cdr x))))
```

Now we use equals for equals Rule of inference to replace e1 by e2 to make the following step in our proof:

```
 (+ 1 (+ 1 (len (dup (cdr x))))
```
={*Use Instantiation of len with ((A (cons (car x) (dup (cdr x)))))*}
```
 (+ 1 (len (cons (car x) (dup (cdr x)))))
```

I skipped a lot of small reasoning steps, but you got the idea right. But the method I showed in class was simpler, so you should stick with that, although once you practice proofs, you will find the above also simple.

Instead of writing

```
 (+ 1 (+ 1 (len (dup (cdr x))))
```
={*Use Instantiation of len with ((A (cons (car x) (dup (cdr x)))))*}
```
 (+ 1 (len (cons (car x) (dup (cdr x))))
```

I could have written:

```
 (+ 1 (+ 1 (len (dup (cdr x))))
```
={*Instantiate def. Axiom of len with the subsitution, ((A (cons (car x) (dup (cdr x))))), then use equals with equals rule of inference using instantiation of axioms consp-cons, if-false and cdr-cons, then using instantiation of equality is symmetric axiom, then again apply equals for equals to obtain*}
```
 (+ 1 (len (cons (car x) (dup (cdr x))))
```
But since we are not machines, we will not be that uncool:p