# Tutorial 5

Chapter 9: ex. 5, 6, 7, 8, 10, 13

Chapter 11: ex. 25, 26, 29, 30, 31, 34, and 35

Function definitions:

```
iter :: Int -> (a -> a) -> (a -> a)
iter n f
  | n > 0      = f . iter (n-1) f
  | otherwise = id

reverse :: [a] → [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

# Tutorial 5

```
elem :: a →  [a] → a
elem  y [] = False
elem  y (x:xs) = if y == x then True else elem y xs

length :: [a] → Int
length [] = 0
length (x:xs) = 1 + length xs

(++) :: [a] → [a] → [a]
(++) [] [] = []
(++) [] (y:ys) = y : ((++) [] ys)
(++) (x:xs) ys = x : ((++) xs ys)
```

# Tutorial 5

```
map f [] = []
map f (x:xs) = f x : map f xs


abs x
    | x < 0 = abs(-x)
    | otherwise = x


signum x
    | x < 0 = -1
    | x == 0 = 0
    | otherwise = 1
```

# Exercise 9.5

to prove for all finite lists `xs` and `ys`:

```
sum (xs ++ ys) = sum xs + sum ys
```

where

```
sum [] = 0
sum (x:xs) = x+sum xs
[] ++ ys = ys
(x:xs)++ys = x:(xs++ys)
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)

= sum (ys)
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)

= sum (ys)

= 0 + sum ys
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)

= sum (ys)

= 0 + sum ys

= sum [] + sum ys
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)
= sum (ys)
= 0 + sum ys
= sum [] + sum ys
```

**Ind:**

```
sum ((x:xs) ++ ys)
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)
= sum (ys)
= 0 + sum ys
= sum [] + sum ys
```

**Ind:**

```
sum ((x:xs) ++ ys)
= sum (x:(xs+ys))
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)

= sum (ys)

= 0 + sum ys

= sum [] + sum ys
```

**Ind:**

```
sum ((x:xs) ++ ys)

= sum (x:(xs+ys))

= x + sum(xs++ys)
```

# Exercise 9.5

**Base:**

```
sum ([] ++ ys)

= sum (ys)

= 0 + sum ys

= sum [] + sum ys
```

**Ind:**

```
sum ((x:xs) ++ ys)

= sum (x:(xs+ys))

= x + sum(xs++ys)

= x + sum xs + sum ys
```

# Exercise 9.5

```
Base:

sum ([] ++ ys)

= sum (ys)

= 0 + sum ys

= sum [] + sum ys

Ind:

sum ((x:xs) ++ ys)

= sum (x:(xs+ys))

= x + sum(xs++ys)

= x + sum xs + sum ys

= sum(x:xs) + sum ys
```

# Exercise 9.6

1. to prove for all finite lists `xs`: `xs ++ [] = xs`
2. to prove for all finite lists `xs`: `(xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

# Exercise 9.6

to prove for all finite lists `xs: xs ++ [] = xs`

**`Base`**

`xs = []: [] ++ [] = []`

# Exercise 9.6

to prove for all finite lists `xs`: `xs ++ [] = xs`

**Base**

`xs = []: [] ++ [] = []`

**Ind.**

`(x:xs) ++ []`

# Exercise 9.6

to prove for all finite lists `xs`: `xs ++ [] = xs`

**Base**

`xs = []`: `[] ++ [] = []`

**Ind.**

```
(x:xs) ++ []

= x:(xs ++ [])
```

# Exercise 9.6

to prove for all finite lists `xs`: `xs ++ [] = xs`

**Base**

`xs = []: [] ++ [] = []`

**Ind.**

`(x:xs) ++ []`

`= x:(xs ++ [])`

`= x:xs`

`QED.`

# Exercise 9.6

to prove for all finite lists `xs`: `(xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**`Base:`**

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

# Exercise 9.6

to prove for all finite lists `xs`: `(xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**`Base:`**

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

= ys ++ zs
```

# Exercise 9.6

to prove for all finite lists `xs`**:** `(xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**`Base:`**

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

= ys ++ zs

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

# Exercise 9.6

to prove for all finite lists `xs: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**`Base:`**

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

= ys ++ zs

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

**`Ind:`**

```
((x:xs) ++ ys) ++ zs
```

# Exercise 9.6

to prove for all finite lists `xs: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**`Base:`**

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

= ys ++ zs

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

**`Ind:`**

```
((x:xs) ++ ys) ++ zs

= (x :~(xs ++ ys)) ++ zs
```

# Exercise 9.6

to prove for all finite lists `xs: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**Base:**

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

= ys ++ zs

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

**Ind:**

```
((x:xs) ++ ys) ++ zs

= (x :~(xs ++ ys)) ++ zs

= x :~((xs ++ ys) ++ zs)
```

# Exercise 9.6

to prove for all finite lists `xs: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**Base:**

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

= ys ++ zs

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

**Ind:**

```
((x:xs) ++ ys) ++ zs

= (x :~(xs ++ ys)) ++ zs

= x :~((xs ++ ys) ++ zs)

= x :~(xs ++ (ys ++ zs))
```

# Exercise 9.6

to prove for all finite lists `xs: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**Base:**

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

**Ind:**

```
((x:xs) ++ ys) ++ zs
= (x :~(xs ++ ys)) ++ zs
= x :~((xs ++ ys) ++ zs)
= x :~(xs ++ (ys ++ zs))
= (x:xs) ++ (ys ++ zs)
```

# Exercise 9.6

to prove for all finite lists `xs: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)`

**Base:**

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

**Ind:**

```
((x:xs) ++ ys) ++ zs
= (x :~(xs ++ ys)) ++ zs
= x :~((xs ++ ys) ++ zs)
= x :~(xs ++ (ys ++ zs))
= (x:xs) ++ (ys ++ zs)
= x :~(xs ++ (ys ++ zs))
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs
2. to prove for all finite lists `xs: length(reverse xs) = length xs`

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

**Base:**

```
xs = []: sum(reverse []) = sum []
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

**`Base:`**

```
xs = []: sum(reverse []) = sum []
```

**`Ind:`**

```
sum (reverse (x:xs))
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

**Base:**

```
xs = []: sum(reverse []) = sum []
```

**Ind:**

```
sum (reverse (x:xs))

= sum (reverse xs ++ [x])     -- def reverse
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

**Base:**

```
xs = []: sum(reverse []) = sum []
```

**Ind:**

```
sum (reverse (x:xs))

= sum (reverse xs ++ [x])     -- def reverse

= sum (reverse xs) + sum [x] -- sum lemma below
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

**Base:**

```
xs = []: sum(reverse []) = sum []
```

**Ind:**

```
sum (reverse (x:xs))

= sum (reverse xs ++ [x])     -- def reverse

= sum (reverse xs) + sum [x] -- sum lemma below

= sum (reverse xs) + x        -- def sum
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

```
Base:

xs = []: sum(reverse []) = sum []

Ind:

sum (reverse (x:xs))

= sum (reverse xs ++ [x])     -- def reverse

= sum (reverse xs) + sum [x] -- sum lemma below

= sum (reverse xs) + x        -- def sum

= x + sum (reverse xs)        -- commutativity +
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

**Base:**

```
xs = []: sum(reverse []) = sum []
```

**Ind:**

```
sum (reverse (x:xs))

= sum (reverse xs ++ [x])     -- def reverse

= sum (reverse xs) + sum [x] -- sum lemma below

= sum (reverse xs) + x        -- def sum

= x + sum (reverse xs)        -- commutativity +

= x + sum xs                  -- inductive hypothesis
```

# Exercise 9.7

1. to prove for all finite lists xs: sum(reverse xs) = sum xs

**Base:**

```
xs = []: sum(reverse []) = sum []
```

**Ind:**

```
sum (reverse (x:xs))
= sum (reverse xs ++ [x])     -- def reverse
= sum (reverse xs) + sum [x]  -- sum lemma below
= sum (reverse xs) + x        -- def sum
= x + sum (reverse xs)        -- commutativity +
= x + sum xs                  -- inductive hypothesis
= sum (x:xs)                  -- definition of sum
```

# Exercise 9.7

```
sum ([] ++ ys)
```

# Exercise 9.7

```
sum ([] ++ ys)
= sum ys                    -- def (++)
```

# Exercise 9.7

```
sum ([] ++ ys)

= sum ys              -- def (++)

= 0 + sum ys          -- identity of addition
```

# Exercise 9.7

```
sum ([] ++ ys)
= sum ys             -- def (++)
= 0 + sum ys         -- identity of addition
= sum [] ++ sum ys -- def sum
```

# Exercise 9.7

```
sum ([] ++ ys)
= sum ys              -- def (++)
= 0 + sum ys          -- identity of addition
= sum [] ++ sum ys -- def sum


sum ((x:xs) ++ ys)
```

# Exercise 9.7

```
sum ([] ++ ys)

= sum ys              -- def (++)

= 0 + sum ys          -- identity of addition

= sum [] ++ sum ys -- def sum


sum ((x:xs) ++ ys)

= sum (x : (xs ++ ys))  -- def (++)
```

# Exercise 9.7

```
sum ([] ++ ys)
= sum ys              -- def (++)
= 0 + sum ys          -- identity of addition
= sum [] ++ sum ys -- def sum


sum ((x:xs) ++ ys)
= sum (x : (xs ++ ys))  -- def (++)
= x + sum (xs ++ ys)    -- def sum
```

# Exercise 9.7

```
sum ([] ++ ys)
= sum ys                -- def (++)
= 0 + sum ys            -- identity of addition
= sum [] ++ sum ys -- def sum


sum ((x:xs) ++ ys)
= sum (x : (xs ++ ys))   -- def (++)
= x + sum (xs ++ ys)     -- def sum
= x + (sum xs + sum ys) -- inductive hypothesis
```

# Exercise 9.7

```
sum ([] ++ ys)
= sum ys              -- def (++)
= 0 + sum ys          -- identity of addition
= sum [] ++ sum ys -- def sum


sum ((x:xs) ++ ys)
= sum (x : (xs ++ ys))  -- def (++)
= x + sum (xs ++ ys)    -- def sum
= x + (sum xs + sum ys) -- inductive hypothesis
= (x + sum xs) + sum ys -- associativity +
```

# Exercise 9.7

```
sum ([] ++ ys)
= sum ys              -- def (++)
= 0 + sum ys          -- identity of addition
= sum [] ++ sum ys -- def sum


sum ((x:xs) ++ ys)
= sum (x : (xs ++ ys))  -- def (++)
= x + sum (xs ++ ys)    -- def sum
= x + (sum xs + sum ys) -- inductive hypothesis
= (x + sum xs) + sum ys -- associativity +
= sum(x:xs) + sum ys
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**Base**

```
xs
= []: length(reverse [])
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**`Base`**

```
xs
= []: length(reverse [])
= length []
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**Base**

```
xs

= []: length(reverse [])

= length []
```

**Ind:**

```
length (reverse (x:xs))
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**Base**

```
xs

= []: length(reverse [])

= length []
```

**Ind:**

```
length (reverse (x:xs))

= length (reverse xs ++ [x])        -- def reverse
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**Base**

```
xs

= []: length(reverse [])

= length []
```

**Ind:**

```
length (reverse (x:xs))

= length (reverse xs ++ [x])        -- def reverse

= length (reverse xs) + length [x] -- length lemma below
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**Base**

```
xs

= []: length(reverse [])

= length []
```

**Ind:**

```
length (reverse (x:xs))

= length (reverse xs ++ [x])        -- def reverse

= length (reverse xs) + length [x] -- length lemma below

= length (reverse xs) + 1           -- def length
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**Base**

```
xs

= []: length(reverse [])

= length []
```

**Ind:**

```
length (reverse (x:xs))

= length (reverse xs ++ [x])        -- def reverse

= length (reverse xs) + length [x] -- length lemma below

= length (reverse xs) + 1          -- def length

= 1 + length (reverse xs)          -- commutativity +
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

```
Base
xs
= []: length(reverse [])
= length []
Ind:
length (reverse (x:xs))
= length (reverse xs ++ [x])        -- def reverse
= length (reverse xs) + length [x]  -- length lemma below
= length (reverse xs) + 1           -- def length
= 1 + length (reverse xs)           -- commutativity +
= 1 + length xs                     -- inductive hypothesis
```

# Exercise 9.7

2. to prove for all finite lists `xs: length(reverse xs) = length xs`

**Base**

```
xs
= []: length(reverse [])
= length []
```

**Ind:**

```
length (reverse (x:xs))
= length (reverse xs ++ [x])          -- def reverse
= length (reverse xs) + length [x]  -- length lemma below
= length (reverse xs) + 1            -- def length
= 1 + length (reverse xs)            -- commutativity +
= 1 + length xs                      -- inductive hypothesis
= length (x:xs)                      -- definition of length
```

# Exercise 9.7

```
length ([] ++ ys)
```

# Exercise 9.7

```
length ([] ++ ys)

= length ys                  -- def (++)
```

# Exercise 9.7

```
length ([] ++ ys)

= length ys                 -- def (++)

= 0 + length ys             -- identity of addition
```

# Exercise 9.7

```
length ([] ++ ys)
= length ys                -- def (++)
= 0 + length ys            -- identity of addition
= length [] ++ length ys -- def length
```

# Exercise 9.7

```
length ([] ++ ys)

= length ys                  -- def (++)

= 0 + length ys              -- identity of addition

= length [] ++ length ys -- def length


length ((x:xs) ++ ys)
```

# Exercise 9.7

```
length ([] ++ ys)
= length ys                 -- def (++)
= 0 + length ys             -- identity of addition
= length [] ++ length ys -- def length

length ((x:xs) ++ ys)
= length (x : (xs ++ ys))      -- def (++)
```

# Exercise 9.7

```
length ([] ++ ys)
= length ys                -- def (++)
= 0 + length ys            -- identity of addition
= length [] ++ length ys -- def length


length ((x:xs) ++ ys)
= length (x : (xs ++ ys))     -- def (++)
= 1 + length (xs ++ ys)       -- def length
```

# Exercise 9.7

```
length ([] ++ ys)
= length ys                 -- def (++)
= 0 + length ys             -- identity of addition
= length [] ++ length ys -- def length


length ((x:xs) ++ ys)
= length (x : (xs ++ ys))     -- def (++)
= 1 + length (xs ++ ys)       -- def length
= 1 + (length xs + length ys) -- inductive hypothesis
```

# Exercise 9.7

```
length ([] ++ ys)
= length ys                 -- def (++)
= 0 + length ys             -- identity of addition
= length [] ++ length ys -- def length

length ((x:xs) ++ ys)
= length (x : (xs ++ ys))      -- def (++)
= 1 + length (xs ++ ys)        -- def length
= 1 + (length xs + length ys) -- inductive hypothesis
= (1 + length xs) + length ys -- associativity +
```

# Exercise 9.7

```
length ([] ++ ys)
= length ys                   -- def (++)
= 0 + length ys               -- identity of addition
= length [] ++ length ys -- def length


length ((x:xs) ++ ys)
= length (x : (xs ++ ys))     -- def (++)
= 1 + length (xs ++ ys)       -- def length
= 1 + (length xs + length ys) -- inductive hypothesis
= (1 + length xs) + length ys -- associativity +
= length(x:xs) + length ys
```

# Exercise 9.8

to prove for all finite lists `xs` and `ys`:

```
elem z (xs ++ ys) = elem z xs || elem z ys
```

# Exercise 9.8

to prove for all finite lists `xs` and `ys`:

```
elem z (xs ++ ys) = elem z xs || elem z ys
```

**Base**

```
xs = []: elem z ([] ++ ys)
```

# Exercise 9.8

to prove for all finite lists `xs` and `ys`:

```
elem z (xs ++ ys) = elem z xs || elem z ys
```

**`Base`**

```
xs = []: elem z ([] ++ ys)

= elem z ys = False || elem z ys
```

# Exercise 9.8

to prove for all finite lists `xs` and `ys`:

```
elem z (xs ++ ys) = elem z xs || elem z ys
```

**Base**

```
xs = []: elem z ([] ++ ys)
= elem z ys = False || elem z ys
= False
```

# Exercise 9.8

to prove for all finite lists `xs` and `ys`:

```
elem z (xs ++ ys) = elem z xs || elem z ys
```

**Base**

```
xs = []: elem z ([] ++ ys)

= elem z ys = False || elem z ys

= False

= elem z [] || elem z ys
```

# Exercise 9.8

**Ind:**

```
elem z (x:xs ++ ys)
```

# Exercise 9.8

**Ind:**

```
elem z (x:xs ++ ys)

= elem z (x:(xs ++ ys))
```

# Exercise 9.8

**Ind:**

```
elem z (x:xs ++ ys)

= elem z (x:(xs ++ ys))

   case z==x:        = True
```

# Exercise 9.8

**Ind:**

```
elem z (x:xs ++ ys)
= elem z (x:(xs ++ ys))
   case z==x:        = True
                     = True || elem z ys
```

# Exercise 9.8

**Ind:**

```
elem z (x:xs ++ ys)
= elem z (x:(xs ++ ys))
   case z==x:       = True
                    = True || elem z ys
                    = elem z (x:xs) || elem z ys
```

# Exercise 9.8

```
Ind:

elem z (x:xs ++ ys)

= elem z (x:(xs ++ ys))

   case z==x:       = True

                    = True || elem z ys

                    = elem z (x:xs) || elem z ys


    case z/=x:      =  elem z (xs ++ ys)
```

# Exercise 9.8

**Ind:**

```
elem z (x:xs ++ ys)
= elem z (x:(xs ++ ys))
   case z==x:       = True
                    = True || elem z ys
                    = elem z (x:xs) || elem z ys

    case z/=x:      =  elem z (xs ++ ys)
                    =  elem z xs || elem z ys
```

# Exercise 9.8

**Ind:**

```
elem z (x:xs ++ ys)
= elem z (x:(xs ++ ys))
   case z==x:       = True
                    = True || elem z ys
                    = elem z (x:xs) || elem z ys

    case z/=x:      =  elem z (xs ++ ys)
                    =  elem z xs || elem z ys
                    =  elem z (x:xs) || elem z ys
```

# Exercise 9.10

to prove for all finite lists `xs` and `defined n`:

```
take n xs ++ drop n xs = xs
```

where

```
take 0 _ = [], take _ [] = [], take n (x:xs)=x:take (n-1) xs
drop 0 xs = xs, drop n [] = [], drop n (x:xs) = drop (n-1) xs
```

# Exercise 9.10

to prove for all finite lists `xs` and `defined n:`

```
take n xs ++ drop n xs = xs
```

where

```
take 0 _ = [], take _ [] = [], take n (x:xs)=x:take (n-1) xs
drop 0 xs = xs, drop n [] = [], drop n (x:xs) = drop (n-1) xs
```

**Base case for** `n==0:`    `take 0 xs ++ drop 0 xs = [] ++ xs = xs`

# Exercise 9.10

to prove for all finite lists `xs` and `defined n:`

```
take n xs ++ drop n xs = xs
```

where

```
take 0 _ = [], take _ [] = [], take n (x:xs)=x:take (n-1) xs
drop 0 xs = xs, drop n [] = [], drop n (x:xs) = drop (n-1) xs
```

**Base case for** `n==0:` `take 0 xs ++ drop 0 xs = [] ++ xs = xs`

**Base case for** `xs==[]:` `take n [] ++ drop n [] = [] ++ [] = []`

# Exercise 9.10

to prove for all finite lists `xs` and `defined n`:

```
take n xs ++ drop n xs = xs
```

where

```
take 0 _ = [], take _ [] = [], take n (x:xs)=x:take (n-1) xs
drop 0 xs = xs, drop n [] = [], drop n (x:xs) = drop (n-1) xs
```

**Base case for** `n==0:     take 0 xs ++ drop 0 xs = [] ++ xs = xs`

**Base case for** `xs==[]:  take n [] ++ drop n [] = [] ++ [] = []`

**Ind.:** `for n > 0,`

```
take n (x:xs) ++ drop n (x:xs)
```

# Exercise 9.10

to prove for all finite lists `xs` and `defined n:`

```
take n xs ++ drop n xs = xs
```

where

```
take 0 _ = [], take _ [] = [], take n (x:xs)=x:take (n-1) xs
drop 0 xs = xs, drop n [] = [], drop n (x:xs) = drop (n-1) xs
```

**`Base case for`** `n==0:`    `take 0 xs ++ drop 0 xs = [] ++ xs= xs`

**`Base case for`** `xs==[]:`  `take n [] ++ drop n [] = [] ++ [] = []`

**`Ind.:`** `for n > 0,`

```
take n (x:xs) ++ drop n (x:xs)
= (x:take (n-1) xs) ++ drop (n-1) xs
```

# Exercise 9.10

to prove for all finite lists `xs` and `defined n:`

```
take n xs ++ drop n xs = xs
```

where

```
take 0 _ = [], take _ [] = [], take n (x:xs)=x:take (n-1) xs
drop 0 xs = xs, drop n [] = [], drop n (x:xs) = drop (n-1) xs
```

**Base case for** `n==0:     take 0 xs ++ drop 0 xs = [] ++ xs = xs`

**Base case for** `xs==[]:  take n [] ++ drop n [] = [] ++ [] = []`

**Ind.:** `for n > 0,`

```
take n (x:xs) ++ drop n (x:xs)
= (x:take (n-1) xs) ++ drop (n-1) xs
= x:(take (n-1) xs ++ drop (n-1) xs)
```

# Exercise 9.10

to prove for all finite lists `xs` and `defined n:`

```
take n xs ++ drop n xs = xs
```

where

```
take 0 _ = [], take _ [] = [], take n (x:xs)=x:take (n-1) xs
drop 0 xs = xs, drop n [] = [], drop n (x:xs) = drop (n-1) xs
```

**Base case for** `n==0:    take 0 xs ++ drop 0 xs = [] ++ xs = xs`

**Base case for** `xs==[]:  take n [] ++ drop n [] = [] ++ [] = []`

**Ind.:** `for n > 0,`

```
take n (x:xs) ++ drop n (x:xs)
= (x:take (n-1) xs) ++ drop (n-1) xs
= x:(take (n-1) xs ++ drop (n-1) xs)
= x:xs
```

# Exercise 9.13

to prove for all natural numbers `n`: `fac2 n = fac n`

```
where fac 0 = 1, fac n = n*fac (n-1)
```

# Exercise 9.13

to prove for all natural numbers `n`: `fac2 n = fac n`

```
where fac 0 = 1, fac n = n*fac (n-1)
```

A direct infuctive proof gets stuck. We need to generalize this to

```
facAux n p = p*fac n
```

This is easy to prove by induction on `n`:

# Exercise 9.13

to prove for all natural numbers `n:` `fac2 n = fac n`

```
where fac 0 = 1, fac n = n*fac (n-1)
```

A direct infuctive proof gets stuck. We need to generalize this to

```
facAux n p = p*fac n
```

This is easy to prove by induction on `n:`

**`Base:`**

```
facAux 0 p
```

# Exercise 9.13

to prove for all natural numbers `n`: `fac2 n = fac n`

```
where fac 0 = 1, fac n = n*fac (n-1)
```

A direct infuctive proof gets stuck. We need to generalize this to

```
facAux n p = p*fac n
```

This is easy to prove by induction on `n`:

**Base:**

```
facAux 0 p
= p = p*1
```

# Exercise 9.13

to prove for all natural numbers `n`: `fac2 n = fac n`

```
where fac 0 = 1, fac n = n*fac (n-1)
```

A direct infuctive proof gets stuck. We need to generalize this to

```
facAux n p = p*fac n
```

This is easy to prove by induction on `n`:

**Base:**

```
facAux 0 p
= p = p*1
= p*fac 0
```

# Exercise 9.13

**Ind:**

```
facAux (n+1) p
= facAux n ((n+1)*p)
= (n+1)*p*(fac n)
p*fac (n+1) = (n+1)*p*(fac n)
```

Now, we can conclude:

```
fac2 n
= facAux n 1
= 1*fac n
= fac n
```

# Exercise 11.25

for all `x` prove: `f.(g.h) x == (f.g).h x`

# Exercise 11.25

for all `x` prove: `f.(g.h) x == (f.g).h x`

Proof:

```
f.(g.h) x
```

# Exercise 11.25

for all `x` prove: `f.(g.h) x == (f.g).h x`

Proof:

```
f.(g.h) x

= f (g.h x)
```

# Exercise 11.25

for all `x` prove: `f.(g.h) x == (f.g).h x`

Proof:

```
f.(g.h) x

= f (g.h x)

= f (g (h x))
```

# Exercise 11.25

for all `x` prove: `f.(g.h) x == (f.g).h x`

Proof:

```
f.(g.h) x

= f (g.h x)

= f (g (h x))

(f.g).h x
```

# Exercise 11.25

for all `x` prove: `f.(g.h) x == (f.g).h x`

Proof:

```
f.(g.h) x

= f (g.h x)

= f (g (h x))

(f.g).h x

= (f.g) (h x)
```

# Exercise 11.25

for all `x` prove: `f.(g.h) x == (f.g).h x`

Proof:

```
f.(g.h) x
= f (g.h x)
= f (g (h x))
(f.g).h x
= (f.g) (h x)
= f (g (h x))  QED.
```

# Exercise 11.26

for all `f` prove: `id.f = f`

# Exercise 11.26

for all `f` prove: `id.f = f`

Proof:

`(id.f) x`

# Exercise 11.26

for all `f` prove: `id.f = f`

Proof:

```
(id.f) x
= id (f x)
```

# Exercise 11.26

for all `f` prove: `id.f = f`

Proof:

```
(id.f) x
= id (f x)
= f x
 QED.
```

# Exercise 11.29

to prove for all natural `n: iter n id = id`

# Exercise 11.29

to prove for all natural `n: iter n id = id`

**Base:**

```
iter 0 id = id    (by def.)
```

# Exercise 11.29

to prove for all natural `n: iter n id = id`

**Base:**

```
iter 0 id = id    (by def.)
```

**Ind.:**

```
iter (n+1) id x
```

# Exercise 11.29

to prove for all natural `n: iter n id = id`

**Base:**

```
iter 0 id = id    (by def.)
```

**Ind.:**

```
iter (n+1) id x
= id . iter n id x
```

# Exercise 11.29

to prove for all natural `n: iter n id = id`

**Base:**

```
iter 0 id = id    (by def.)
```

**Ind.:**

```
iter (n+1) id x
= id . iter n id x
= id.id x
```

# Exercise 11.29

to prove for all natural `n: iter n id = id`

**Base:**

```
iter 0 id = id    (by def.)
```

**Ind.:**

```
iter (n+1) id x
= id . iter n id x
= id.id x
= id (idx x))
```

# Exercise 11.29

to prove for all natural `n: iter n id = id`

**Base:**

```
iter 0 id = id    (by def.)
```

**Ind.:**

```
iter (n+1) id x
= id . iter n id x
= id.id x
= id (idx x))
= id x
QED.
```

# Exercise 11.30

to prove: `abs.abs x = abs x`

to prove: `signum.signum x = signum x`

# Exercise 11.30

to prove: `abs.abs x = abs x`

# Exercise 11.30

to prove: `abs.abs x = abs x`

```
abs.abs x = abs(abs x)
```

# Exercise 11.30

to prove: `abs.abs x = abs x`

```
abs.abs x = abs(abs x)

    1) case x < 0: abs(abs x)=abs (-x)
```

# Exercise 11.30

to prove: `abs.abs x = abs x`

```
abs.abs x = abs(abs x)

    1) case x < 0: abs(abs x)=abs (-x)

    2) case x >= 0: abs(abs x)=abs x      Q.E.D.
```

# Exercise 11.30

to prove: `abs.abs x = abs x`

```
abs.abs x = abs(abs x)

    1) case x < 0: abs(abs x)=abs (-x)

    2) case x >= 0: abs(abs x)=abs x     Q.E.D.
```

to prove: `signum.signum x = signum x`

# Exercise 11.30

to prove: `abs.abs x = abs x`

```
abs.abs x = abs(abs x)

    1) case x < 0: abs(abs x)=abs (-x)

    2) case x >= 0: abs(abs x)=abs x     Q.E.D.
```

to prove: `signum.signum x = signum x`

```
    1) case x==0:  signum.signum 0=signum (signum 0) = signum0
```

# Exercise 11.30

to prove: `abs.abs x = abs x`

```
abs.abs x = abs(abs x)

    1) case x < 0: abs(abs x)=abs (-x)

    2) case x >= 0: abs(abs x)=abs x     Q.E.D.
```

to prove: `signum.signum x = signum x`

```
    1) case x==0:  signum.signum 0=signum (signum 0) = signum0

    2) case x>0:   signum.signum x=signum 1 = 1 = signum x
```

# Exercise 11.30

to prove: `abs.abs x = abs x`

```
abs.abs x = abs(abs x)
    1) case x < 0: abs(abs x)=abs (-x)
    2) case x >= 0: abs(abs x)=abs x     Q.E.D.
```

to prove: `signum.signum x = signum x`

```
    1) case x==0:  signum.signum 0=signum (signum 0) = signum0
    2) case x>0:   signum.signum x=signum 1 = 1 = signum x
    3) case x<0:   signum.signum x=signum (-1) = -1 = signum x
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**`Base:`**

```
map f ([]++zs)
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**Base:**

```
map f ([]++zs)
= map f zs
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**`Base:`**

```
map f ([]++zs)
= map f zs
= [] ++ map f zs
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**`Base:`**

```
map f ([]++zs)

= map f zs

= [] ++ map f zs

= map f [] ++ map f zs
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**Base:**

```
map f ([]++zs)
= map f zs
= [] ++ map f zs
= map f [] ++ map f zs
```

**Ind.:**

```
map f ((y:ys)++zs)
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**`Base:`**

```
map f ([]++zs)

= map f zs

= [] ++ map f zs

= map f [] ++ map f zs
```

**`Ind.:`**

```
map f ((y:ys)++zs)

= f (y:(ys++zs)
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**Base:**

```
map f ([]++zs)

= map f zs

= [] ++ map f zs

= map f [] ++ map f zs
```

**Ind.:**

```
map f ((y:ys)++zs)

= f (y:(ys++zs)

= f y: map f (ys++zs)
```

# Exercise 11.31

to prove: `map f (ys++zs) = map f ys ++ map f zs`

**Base:**

```
map f ([]++zs)

= map f zs

= [] ++ map f zs

= map f [] ++ map f zs
```

**Ind.:**

```
map f ((y:ys)++zs)

= f (y:(ys++zs)

= f y: map f (ys++zs)

= f y: (map f ys ++ map f zs)
```

# Exercise 11.31

```
map f (y:ys) ++ map f zs
```

# Exercise 11.31

```
map f (y:ys) ++ map f zs

= (f y: (map f ys)) ++ map f zs
```

# Exercise 11.31

```
map f (y:ys) ++ map f zs
= (f y: (map f ys)) ++ map f zs
= f y : (map f ys ++ map f zs)
QED.
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []

map f (concat []) = map f [] = []
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**`Base:`**

```
concat (map (map f) []) = concat [] = []
map f (concat []) = map f [] = []
```

**`Ind.:`**

```
concat (map (map f) (x:xs))
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []
map f (concat []) = map f [] = []
```

**Ind.:**

```
concat (map (map f) (x:xs))
= concat (((map f) x):map (map f) xs)
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []
map f (concat []) = map f [] = []
```

**Ind.:**

```
concat (map (map f) (x:xs))
= concat (((map f) x):map (map f) xs)
= concat ((map f x):map (map f) xs)
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []
map f (concat []) = map f [] = []
```

**Ind.:**

```
concat (map (map f) (x:xs))
= concat (((map f) x):map (map f) xs)
= concat ((map f x):map (map f) xs)
= (map f x) ++ concat (map (map f) xs)
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []

map f (concat []) = map f [] = []
```

**Ind.:**

```
concat (map (map f) (x:xs))

= concat (((map f) x):map (map f) xs)

= concat ((map f x):map (map f) xs)

= (map f x) ++ concat (map (map f) xs)

= (map f x) ++ map f (concat xs)
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []

map f (concat []) = map f [] = []
```

**Ind.:**

```
concat (map (map f) (x:xs))

= concat (((map f) x):map (map f) xs)

= concat ((map f x):map (map f) xs)

= (map f x) ++ concat (map (map f) xs)

= (map f x) ++ map f (concat xs)

map f (concat (x:xs))
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []

map f (concat []) = map f [] = []
```

**Ind.:**

```
concat (map (map f) (x:xs))

= concat (((map f) x):map (map f) xs)

= concat ((map f x):map (map f) xs)

= (map f x) ++ concat (map (map f) xs)

= (map f x) ++ map f (concat xs)

map f (concat (x:xs))

= map f (x ++ concat xs)
```

# Exercise 11.34

to prove: `concat (map (map f) xs) = map f (concat xs)`

**Base:**

```
concat (map (map f) []) = concat [] = []
map f (concat []) = map f [] = []
```

**Ind.:**

```
concat (map (map f) (x:xs))
= concat (((map f) x):map (map f) xs)
= concat ((map f x):map (map f) xs)
= (map f x) ++ concat (map (map f) xs)
= (map f x) ++ map f (concat xs)
map f (concat (x:xs))
= map f (x ++ concat xs)
= (map f x) ++ map f (concat xs)
```

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

Proof:

```
((0<) . (+1)) x = (0<) ((+1) x)
```

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

Proof:

```
((0<) . (+1)) x = (0<) ((+1) x)
= (0<) ((\a -> a+1) x) = (0<)(x+1)
```

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

Proof:

```
((0<) . (+1)) x = (0<) ((+1) x)
= (0<) ((\a -> a+1) x) = (0<)(x+1)
= (\a -> 0<a)(x+1) = 0<x+1
```

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

Proof:

```
((0<) . (+1)) x = (0<) ((+1) x)

= (0<) ((\a -> a+1) x) = (0<)(x+1)

= (\a -> 0<a)(x+1) = 0<x+1


(0<=)x
```

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

Proof:

```
((0<) . (+1)) x = (0<) ((+1) x)

= (0<) ((\a -> a+1) x) = (0<)(x+1)

= (\a -> 0<a)(x+1) = 0<x+1
```

```
(0<=)x

= (\a -> 0<=a)x
```

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

Proof:

```
((0<) . (+1)) x = (0<) ((+1) x)
= (0<) ((\a -> a+1) x) = (0<)(x+1)
= (\a -> 0<a)(x+1) = 0<x+1


(0<=)x
= (\a -> 0<=a)x
= 0<=x
```

# Exercise 11.35

to prove: `(0<) . (+1) = (0<=)`

Proof:

```
((0<) . (+1)) x = (0<) ((+1) x)
= (0<) ((\a -> a+1) x) = (0<)(x+1)
= (\a -> 0<a)(x+1) = 0<x+1
```

```
(0<=)x
= (\a -> 0<=a)x
= 0<=x
= 0<x+1
```