Exercise 3.1.
Show that the following method satisfies total correctness by supplying a weakest precondition proof and termination metric.

```
method Mult(x: int, y: int) returns (r: int)
      requires 0 <= x && 0 <= y
      ensures r == x * y
{
      if x == 0 {
            r := 0;
      } else {
            var z := Mult(x - 1, y);
            r := z + y;
      }
}
```

In the solution below, references are included to laws used from Appendix A of Programming from Specifications by Carroll Morgan, e.g., (A.22) refers to law A.22.

```
method Mult(x: int, y: int) returns (r: int)
      requires 0 <= x && 0 <= y
      ensures r == x * y
      decreases x
{
      { x == 0 || 1 <= x && 0 <= y }                                          (A.22)
      { true && (x != 0 ==> 1 <= x && 0 <= y) }
      { (x == 0 ==> 0 == x * y) && (x != 0 ==> 1 <= x && 0 <= y) }
      if x == 0 {
            { 0 == x * y }
            r := 0;
            { r == x * y }
      } else {
            { 1 <= x && 0 <= y }
            { 0 <= x - 1 && 0 <= y && x * y == x * y }
            { 0 <= x - 1 && 0 <= y && (x - 1) * y + y == x * y }   one-point rule (A.56)
            { 0 <= x - 1 && 0 <= y && forall r' :: r' == (x - 1) * y ==> r' + y == x * y }
                                                                              (A.74)
            { forall z :: 0 <= x - 1 && 0 <= y &&
                        forall r' :: r' == (x - 1) * y ==> r' + y == x * y }
            var z := Mult(x - 1, y);
            { z + y == x * y }
            r := z + y;
            { r == x * y }
      }
      { r == x * y }
}
```

The method is correct since the given precondition, 0 <= x && 0 <= y implies the calculated weakest precondition, x == 0 || 1 <= x && 0 <= y.

Exercise 3.2.
Write a simple `decreases` clause (i.e., without using a lexiographic tuple) that proves termination for each of the following functions. (You saw some of these examples in the lectures this week. Attempt them again to confirm your understanding of their solutions.)

```
a) function F(x: int): int {
       if x < 10 then x else F(x - 1)
   }
```

decreases x

```
b) function G(x: int): int {
        if x >= 0 then G(x - 2) else x
   }
```

decreases x

```
c) function H(x: int): int {
       if x < -60 then x else H(x - 1)
   }
```

decreases x + 60

```
d) function I(x: nat, y: nat): int {
       if x == 0 || y == 0 then
            12
       else if x % 2 == y % 2 then
            I(x - 1, y)
       else
            I(x, y - 1)
    }
```

decreases x + y

When x % 2 == y % 2, we have x + y > x – 1 + y.

When x % 2 != y % 2, we have x + y > x + y – 1.

```
e)  function M(x: int, b: bool): int {
        if b then x else M(x + 25, true)
    }
```

decreases !b

There is no need to use x in the termination metric.

```
f)  function L(x: int): int {
        if x < 100 then L(x + 1) + 10 else x
    }
```

decreases 100 – x

In this case, x increases so we want to find a termination metric which decreases when x increases, and that is not negative when a recursive call is made.

g)
```
function J(x: nat, y: nat): int {

    if x == 0 then
         y
    else if y == 0 then
         J(x - 1, 3)
    else
         J(x, y - 1)
 }
```

decreases 4 * x + y

When y ==0, we have 4 * x + 0 > 4 * (x – 1) + 3. That is, 4 * x > 4 * x – 1.

When y != 0, we have 4 * x + y > 4 * x + y – 1.

Exercise 3.3.
Determine if the first tuple exceeds the second. (You saw some of these examples in the lectures this week. Attempt them again to confirm your understanding of their solutions.)

a) `2,5 and 1,7`

Yes.

b) `1,7 and 7,1`

No.

c) `5,0,8 and 4,93`

Yes.

d) `4,9,3 and 4,93`

No.

e) `4,93 and 4,9,3`

Yes.

f) `3 and 2,9`

Yes.

g) `true,80 and false,66`

Yes.

h) `true,2 and 19,1`

No.

i) `4,true,50 and 4,false,800`

Yes.

j) 7.0,{3,4,9},false,10 and 7.0,{3,9},true,10

Yes.

Exercise 3.4.
Recall the method `Study` from the lectures and repeated below.

```
method Study(n: nat, h: nat)
  decreases n, h
{
  if h != 0 {
    Study(n, h - 1);
  } else if n == 0 {
  } else {
    var hours := RequiredStudyTime(n - 1);
    Study(n - 1, hours);
  }
}
```

Suppose the university cracks down on professors to limit the required study time for a course to 200 hours. We can then write a postcondition for method `RequiredStudyTime`:

```
method RequiredStudyTime(c: nat) returns (hours: nat)
  ensures hours <= 200
```

Knowing such a bound, you can still use the termination metric `n,h` for `Study`, but it's also possible to write a termination metric without using a lexicographic tuple. How?

decreases n*201 + h

When h != 0, we have n*201 + h > n*201 + h -1.

When h == 0 (and n != 0), we have n*201 + 0 > (n-1)*201 + hours. That is, n*201 > n*201 – 201 + hours, where hours <= 200.

Exercise 3.5.
Recall the program `StudyPlan` from the lectures and repeated below.

```
method StudyPlan(n: nat)
  requires n <= 40
  decreases 40 - n
{
  if n == 40 { // done }
  else {
      var hours := RequiredStudyTime(n);
      Learn(n, RequiredStudyTime(n));
  }
}

method Learn(n: nat, h: nat)
  requires n < 40
  decreases 40 - n, h
{
  if h == 0 { StudyPlan(n + 1); } else { Learn(n, h - 1); }
}
```

Add `decreases` clauses to prove termination of the following variations of `StudyPlan` written in terms of an `Outer` and `Inner` method.

```
a) method Outer(a: nat) {
      if a != 0 {
           var b := RequiredStudyTime(a - 1);
           Inner(a, b);
      }
   }

   method Inner(a: nat, b: nat)
       requires a != 0
  {
      if b == 0 {
           Outer(a - 1);
      } else {
           Inner(a, b - 1);
      }
   }
```

Outer: decreases a
Inner: decreases a, b

When Inner called from Outer, we have a > a, b.
When Outer called from Inner, we have a, b > a-1.
When Inner called from Inner, we have a, b > a, b-1.

The call to RequiredStudyTime is not a recursive call and does not require a decrease in the termination metric.

```
b) method Outer(a: nat) {
       if a != 0 {
            var b := RequiredStudyTime(a - 1);
            Inner(a - 1, b);
       }
    }

    method Inner(a: nat, b: nat) {
       if b == 0 {
          Outer(a);
      } else {
          Inner(a, b - 1);
      }
    }
```

Outer: decreases a, 0
Inner: decreases a, 1, b

When Inner called from Outer, we have a, 0 > a-1, 1, b.
When Outer called from Inner, we have a, 1, b > a, 0.
When Inner called from Inner, we have a, 1, b > a, 1, b-1.

Exercise 3.6.
The `StudyPlan` example in the lectures (see Exercise 3.5 above) benefitted from the rule that a proper prefix exceeds a longer lexicographic tuple. Write `decreases` clauses for

`StudyPlan` and `Learn` that prove termination without making use of that rule. That is, your decreases clauses for `StudyPlan` and `Learn` will be equally long.

Study plan: decreases 40-n, 1, 0
Learn: decreases 40-n, 0, h

When StudyPlan called from Learn, we have 40-n, 0, h > 40-(n+1), 1, 0.
When Learn called from StudyPlan, we have 40-n, 1, 0 > 40-n, 0, RequiredStudyTime(n).
When Learn called from Learn, we have 40-n, 0, h > 40-n, 0, h-1.