Ex. 5.3, pg. 371 textbook (also ex. 5.1.1, pg. 204 exercisebook)

Prove that the following program calculates sqrt = floor(n^(1/2)):

```
{n>1}
begin
  i:= 1;
  z:= 1;
  while z <= n do
    i:= i+1;
    z:= i*i;
  od
  sqrt := i-1;
end
{sqrt² <= n and n < (sqrt+1)²}
```

According to the composition rule IR1, we can split the proof in three steps:

1. `{n>1} i:= 1; z:= 1; {I}`
2. `{I} while . . . od {I and z>n}`
3. `{I and z>n} sqrt := i-1; {sqrt² <= n and n < (sqrt+1)²}`

Choose a loop invariant

I = `{z = i² and "something related to the size of i wrt n"}`

The loop terminates when `i = sqrt(n) + 1`, thus it could be reasonable to set:

I = `{z = i² and (i-1)² <= n}`

Proof of point 1

By trivially applying backward substitution, we get:

```
{1=1² and (1-1)² <= n}
i:= 1;
{1=i² and (i-1)² <= n}
z:= 1;
{z = i² and (i-1)² <= n}
```

`{1=1² and (1-1)² <= n}` is equivalent to `{0 <= n}`, which is guaranteed by precondition `{n > 1}`.

Proof of point 2

According to rule IR4 we have to prove that:

`{I and z <= n } = {z = i² and (i-1)² <= n and z <= n}`

```
i:= i+1;
z:= i*i;
{z = i² and (i-1)² <= n}
```

By backward substitution, we get:

`{(i+1-1)² <= n} == {i² <= n}`, which is implied by `{z = i² and z <= n}`

```
i:= i+1;
{i*i = i² and (i-1)² <= n} == {(i-1)² <= n}
z:= i*i;
{z = i² and (i-1)² <= n}
```

Proof of point 3

By backward substitution, we get:
$\{(i-1)^2 <= n \text{ and } n < (i-1+1)^2\} == \{(i-1)^2 <= n \text{ and } n < i^2\}$, which is implied by $\{I \text{ and } z>n\} == \{z = i^2 \text{ and } (i-1)^2 <= n \text{ and } z>n\}$
`sqrt := i-1;`
$\{sqrt^2 <= n \text{ and } n < (sqrt+1)^2\}$

What if the precondition is {n >= 0}?

The new precondition allows the cases n=0 and n=1, previously excluded.

The behavior of the program with n = 0: the loop is not entered, i is not incremented, and sqrt is set to i-1 = 1-1 = 0, which is correct, since sqrt(0) = 0.

The behavior of the program with n = 1: the loop is executed once, i is incremented once, and sqrt is set to i-1 = 2-1 = 1, which is correct, since sqrt(1) = 1.

Thus, we should be able to prove the theorem with the new precondition.

Only the prove of point 1 has to be reconsidered.

Proof of point 1

By trivially applying backward substitution, we get:

$\{1=1^2 \text{ and } (1-1)^2 <= n\} == \{0 <= n\}$, which is exactly the precondition.
`i:= 1;`
$\{1=i^2 \text{ and } (i-1)^2 <= n\}$
`z:= 1;`
$\{z = i^2 \text{ and } (i-1)^2 <= n\}$

Ex. 5.4, pg. 371 textbook

Disprove (i.e. prove wrong):

```
{n>=0}
begin
  x:= 0;
  y:= 1;
  z:= 1;
  while y < n do
    x:= x+1;
    z:= z+2;
    y:= y+z;
  od
  sqrt := x;
end
{sqrt² <= n and n < (sqrt+1)²}
```

This program is based on the property that $(x+1)^2 = x^2+2x+1$. Variable y is meant to store x^2, z is meant to store $2x+1$.

In practice it is easy to find that y stores $(x+1)^2$, not x^2.

In order to disprove the above specification, we need to show a counterexample. Let `n = 1`. The loop body is not executed, and `sqrt = 0` at the end of the computation. However, `0 <= 1`, but `1 = (0 + 1)²`, so the postcondition is made false and the specification is disproved.

In order to see how the application of Hoare's method can suggest where the problem in the program is and also suggest modifications to get a correct result, we try to apply it to the original program and see where things go wrong.

According to the composition rule IR1, we can split the proof in three steps:

1. `{n>=0} x:= 0; y:= 1; z:= 1; {I}`
2. `{I} while . . . od {I and y >= n}`
3. `{I and y >= n} sqrt := x; {sqrt² <= n and n < (sqrt+1)²}`

Choose a loop invariant

I = `{y = (x+1)² and "something related to the size of x wrt n"}`

The loop terminates when x = sqrt(n) , thus it could be reasonable to set:

I = `{y = (x+1)² and x² <= n and z = 2x+1}`

Proof of point 1

By trivially applying backward substitution, we get:

`{1 = 1² and 0 <= n and 1=1}` == `{0 <= n}`, which is the precondition.

```
x:= 0;
y:= 1;
z:= 1;
{y = (x+1)² and x² <= n and z = 2x+1}
```

Proof of point 2

According to rule IR4 we have to prove that:

```
{I and y < n} ==  {y = (x+1)² and x² <= n and z = 2x+1 and y < n}
x:= x+1;
```

```
z:= z+2;
y:= y+z;
{y = (x+1)² and x² <= n and z = 2x+1}
```

By backward substitution:

```
{y+z+2 = (x+2)² and (x+1)² <= n and z+2 = 2x+2+1} ==
{y+2x+1+2 = (x+2)² and (x+1)² <= n and z = 2x+1} ==
{y = x²+2x+1 and (x+1)² <= n and z = 2x+1} ==
{y = (x+1)² and (x+1)² <= n and z = 2x+1},
```

which is implied by `{I and y < n}` since `{y = (x+1)² and y < n}` implies `{(x+1)² < n}`, which implies `(x+1)² <= n.`

```
x:= x+1;
{y+z+2 = (x+1)² and x² <= n and z+2 = 2x+1}
z:= z+2;
{y+z = (x+1)² and x² <= n and z = 2x+1}
y:= y+z;
{y = (x+1)² and x² <= n and z = 2x+1}
```

Proof of point 3

By backward substitution, we get:

`{x² <= n and n < (x+1)²}`, which is *not* implied by `{I and y >= n} == {y = (x+1)² and x² <= n and z = 2x+1 and y >= n}`, which only states that `(x+1)² >= n` (i.e., equal sign could hold).

```
sqrt := x;
{sqrt² <= n and n < (sqrt+1)² }
```

So, we understand that the algorithm does not work properly when n is a perfect square (e.g., for n=1 returns 0 as seen in the counterexample, for n = 4 returns 1, etc.).

Now, let's try to change the program so that it reflects correctly the specification. We try to modify the condition of the loop, which causes the problem connected with proving point 3.

Intuitively, the fact that the program yields incorrect results when the input is a square, is due to the loop being exited too soon. Thus, we modify the condition into y <= n, thus getting one additional iteration when y (i.e., $(x+1)^2$ is exactly equal to n).

Prove:

```
{n>=0}
begin
  x:= 0;
  y:= 1;
  z:= 1;
  while y <= n do
    x:= x+1;
    z:= z+2;
    y:= y+z;
  od
  sqrt := x;
end
{sqrt² <= n and n < (sqrt+1)²}
```

Choose a loop invariant

The loop terminates when x = sqrt(n) , thus it could be reasonable to set:

I = `{y = (x+1)² and x² <= n and z = 2x+1}`

Proof of point 1

By trivially applying backward substitution, we get (*exactly as before*):
{1 = 1² and 0 <= n and 1=1} == {0 <= n}, which is the precondition.
x:= 0;
y:= 1;
z:= 1;
{y = (x+1)² and x² <= n and z = 2x+1}

Proof of point 2

According to rule IR4 we have to prove that:
{I and y <= n} == {y = (x+1)² and x² <= n and z = 2x+1 and y <= n}
x:= x+1;
z:= z+2;
y:= y+z;
{y = (x+1)² and x² <= n and z = 2x+1}

By backward substitution:
{y+z+2 = (x+2)² and (x+1)² <= n and z+2 = 2x+2+1} ==
{y+2x+1+2 = (x+2)² and (x+1)² <= n and z = 2x+1} ==
{y = x²+2x+1 and (x+1)² <= n and z = 2x+1} ==
{y = (x+1)² and (x+1)² <= n and z = 2x+1} == {I and y <= n} == {y = (x+1)² and y <= n and z = 2x+1}.
x:= x+1;
{y+z+2 = (x+1)² and x² <= n and z+2 = 2x+1}
z:= z+2;
{y+z = (x+1)² and x² <= n and z = 2x+1}
y:= y+z;
{y = (x+1)² and x² <= n and z = 2x+1}

Proof of point 3

By backward substitution, we get:
{x² <= n and n < (x+1)²}, which is *the same as* {I and y > n} == {y = (x+1)² and x² <= n and z = 2x+1 and y > n}.
sqrt := x;
{sqrt² <= n and n < (sqrt+1)²}

Ex. 5.1.2, pg. 205 exercisebook

```
lcm:  begin z:=1;
            while z mod x != 0 or z mod y != 0 do
                  z:= z+1;
            od
      end
```

Prove the partial correctness of program **lcm** w.r.t. the following predicates:

Pre: `{x >= 1 and y >= 1}`

Post: `{z mod x = 0 and z mod y = 0 and forall w (1 <= w < z => w mod x != 0 or w mod y != 0)`

That is, {Pre} lcm {Post}

Choose a loop invariant

I: `forall w (1 <= w < z => w mod x != 0 or w mod y != 0)`

According to the composition rule IR1, we can split the proof in three steps:

```
1. {Pre} z:= 1; {I}
2. {I} while . . . od {I and not (z mod x != 0 or z mod y != 0)}
3. {I and not (z mod x != 0 or z mod y != 0)} => {Post}
```

Proof of point 1

By trivially applying backward substitution, we get:
`{forall w (1 <= w < 1 => w mod x != 0 or w mod y != 0)} = forall w (false => …)} ≡ {true}`, which is a more general condition than `{Pre}`

```
z:= 1;
{forall w (1 <= w < z => w mod x != 0 or w mod y != 0)}
```

Proof of point 2

According to rule IR4 we have to prove that:

```
{I and (z mod x != 0 or z mod y != 0)}
z:=z+1;
{I}
```

By backward substitution (notice we replace < by <=):

```
I' = {forall w (1 <= w <= z => w mod x != 0 or w mod y != 0)}
z:=z+1;
{forall w (1 <= w < z => w mod x != 0 or w mod y != 0)}
```

Now consider:

```
I and c = {forall w (1 <= w < z => w mod x != 0 or w mod y != 0)} and
(z mod x != 0 or z mod y != 0)} => I' = {forall w (1 <= w <= z => w
mod x != 0 or w mod y != 0)}
```

, since when `w = z` we get

```
{(z mod x != 0 or z mod y != 0)} => {true => (z mod x != 0 or z mod y
!= 0)}
```

Proof of point 3

Obvious, since `{I and not c} == {Post}`

Ex. 5.1.3 pg. 206 exercisebook

Prove that the following program correctly checks if x is prime (preconditions and postconditions are given).

```
PRIME: {x > 1}
      begin
            i := 2;
            pr := 1;
            while i < x do
                  if x mod i = 0
                        then pr := 0
                  fi
                  i := i + 1
            od
      end
      {(pr = 0 => exists y(1 < y < x and x mod y = 0)) and
       (pr = 1 => forall y(1 < y < x => x mod y != 0)}
```

Choice of loop invariant

The loop invariant is suggested immediately by the postcondition and the loop condition:

I = `i <= x and (pr = 0 => exists y(1 < y < i and x mod y = 0)) and (pr = 1 => forall y(1 < y < i => x mod y != 0)`

Substeps of the (partial) correctness proof

According to IR1, we can split the proof (as usual) into three sequential substeps:

1. `{Pre} i := 2; pr := 1; {I}`
2. `{I} while ... od {I and i >= x}`
3. `{I and i >= x} => {Post}`

Proof of step 1

We backsubstitute twice and get:

`{2 <= x and forall y(1 < y < 2 => x mod y != 0)}` =(y is integer so 1 < y < 2 is always false)= `{x >= 2} == {Pre}`

```
i := 2;
```

```
{i <= x and (1 = 0 => exists y(1 < y < i and x mod y = 0)) and (1 = 1 => forall y(1 < y < i => x mod y != 0)} == {i <= x and forall y(1 < y < i => x mod y != 0)}
```

```
pr := 1;
```

```
{i <= x and (pr = 0 => exists y(1 < y < i and x mod y = 0)) and (pr = 1 => forall y(1 < y < i => x mod y != 0)}
```

Proof of step 2

Using IR4 to handle the while loop, we want to prove:

```
{I and i < x}
if x mod i = 0
  then pr := 0 fi
i := i + 1
{I}
```

We backsubstitute once and get to the new goal:

```
{I and i < x}
if x mod i = 0
  then pr := 0 fi
{i+1 <= x and (pr = 0 => exists y(1 < y < i+1 and x mod y = 0)) and (pr = 1 => forall y(1 < y < i+1 => x mod y != 0)} == {i < x and (pr = 0 => exists y(1 < y <= i and x mod y = 0)) and (pr = 1 => forall y(1 < y <= i => x mod y != 0)} == {Q}
{Q}
i := i + 1
{I}
```

Now we apply IR3b to backsubstitute through the if construct. Therefore we have to prove (Q has been defined above):

1. `{I and i < x and x mod i != 0} => {Q}`
2. `{I and i < x and x mod i = 0} pr := 0 {Q}`

Step 2.1 is proved by noting that:

```
{I and i < x and x mod i != 0} == {i <= x and (pr = 0 => exists y(1 < y < i and x mod y = 0)) and (pr = 1 => forall y(1 < y < i => x mod y != 0) and i < x and x mod i != 0} == {i < x and (pr = 0 => exists y(1 < y < i and x mod y = 0)) and (pr = 1 => forall y(1 < y < i => x mod y != 0) and x mod i != 0}
```

Now, notice that:

- `i < x` is in both the antecedent and in the consequent
- `if pr = 0: exists y(1 < y < i and i mod y = 0) =>` (*a fortiori*) `exists y(1 < y <= i and x mod y = 0)`
- `if pr = 1: exists y(1 < y < i => x mod y != 0) and x mod i != 0 => exists y(1 < y <= i => x mod y != 0) == {Q}`

Thus, the proof of substep 2.1 is completed. Now, for substep 2.2, we backsubstitute:
`{i < x and exists y(1 < y <= i and x mod y = 0)} = {R}`

```
pr := 0
{i < x and (pr = 0 => exists y(1 < y <= i and x mod y = 0)) and (pr = 1 => forall y(1 < y <= i => x mod y != 0)}
```

Now we just need to notice that:

```
{I and i < x and x mod i = 0} == {i <= x and (pr = 0 => exists y(1 < y < i and x mod y = 0)) and (pr = 1 => forall y(1 < y < i => x mod y != 0) and i < x and x mod i = 0} == {i < x and (pr = 0 => exists y
```

```
(1 < y < i and x mod y = 0)) and (pr = 1 => forall y(1 < y < i => x
mod y != 0) and x mod i = 0} => {i < x and exists y(y = i and x mod y
= 0)} => {R}
```

Thus completing step 2.2 and the whole step 2.

Proof of step 3

`{I and i >= x} == {i >= x and i <= x and (pr = 0 => exists y(1 < y < i and x mod y = 0)) and (pr = 1 => forall y(1 < y < i => x mod y != 0)} == {i = x and (pr = 0 => exists y(1 < y < x and x mod y = 0)) and (pr = 1 => forall y(1 < y < x => x mod y != 0))}`, which clearly subsumes the postcondition (just drop the first term of the conjunction).

INFERENCE RULES:

A1: $\{P_x^t\}$ `x := t` $\{P\}$

IR1: $\{P\}S_1\{R\}$, $\{R\}S_2\{Q\}$

|----------

$\{P\}S_1; S_2\{Q\}$

IR2: $P_1 \Rightarrow P$, $\{P\}S\{Q\}$, $Q \Rightarrow Q_1$

|----------

$\{P_1\}S\{Q_1\}$

IR3: $\{P \text{ and } c\}S_1\{Q\}$, $\{P \text{ and } !c\}S_2\{Q\}$

|----------

$\{P\}$ `if c then` S_1 `else` S_2 `fi` $\{Q\}$

IR3b: $\{P \text{ and } c\}S_1\{Q\}$, $P \text{ and } !c \Rightarrow Q$

|----------

$\{P\}$ `if c then` S_1 `fi` $\{Q\}$

IR4: $\{I \text{ and } c\}S\{I\}$

|----------

$\{I\}$ `while c do S od` $\{I \text{ and } !c\}$