CSE 331
Software Design & Implementation

Autumn 2023

Section 6 – Imperative Programming I

Administrivia

- HW6 released later today
  - Due Wednesday (11/8) @ 11:00pm

Hoare Triples – Review

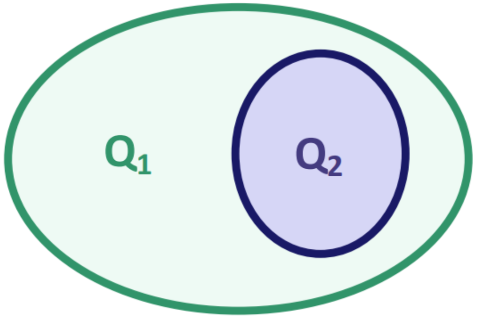
- A Hoare Triple has 2 assertions and some code

  {{ P }}
  S
  {{ Q }}
  - P is a precondition, Q is the postcondition
  - S is the code

- Triple is “valid” if the code is correct:
  - S takes any state satisfying P into a state satisfying Q
    - Does not matter what the code does if P does not hold initially

Stronger vs Weaker – Review

- Assertion is stronger iff it holds in a subset of states
  - **Stronger** assertion implies the **weaker** one:
    If Q_2 is true, Q_1 must also be true, $Q_2 \rightarrow Q_1$

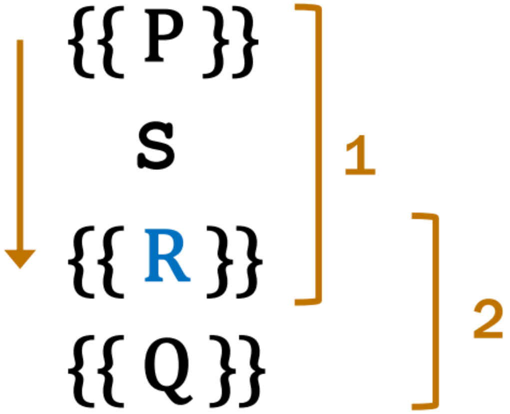


- Different from strength in *specifications*:
  - A stronger spec:
    - Stronger postcondition: guarantees more specific output
    - Weaker precondition: handles more allowable inputs

  compared to a weaker one

Forward Reasoning – Review

- Forwards reasoning fills in the postcondition
  - Gives strongest postcondition making the triple valid
- Apply forward reasoning to fill in R



{{ P }}

S

{{ R }}

{{ Q }}

  - Check second triple by proving that R implies Q

Question 1a

$\{\{\, x \geq 3 \,\}\}$

`y = x - 2;`

$\{\{$ ______________________________ $\}\}$

`z = 2 * y;`

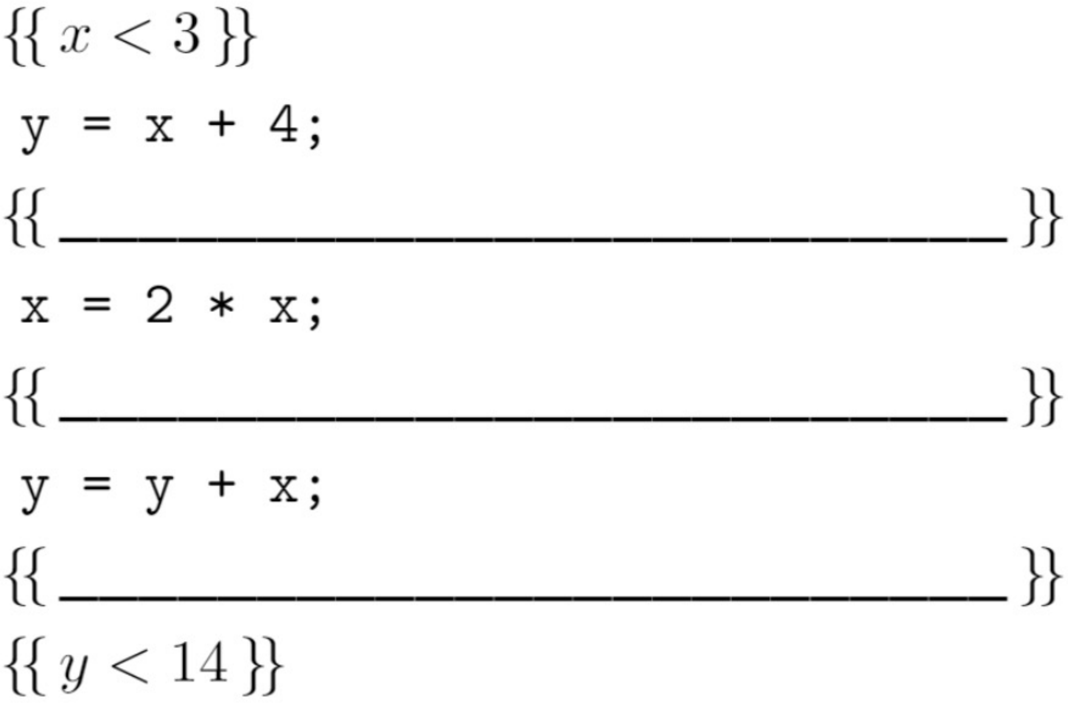
$\{\{$ ______________________________ $\}\}$

`z = z - 2;`

$\{\{$ ______________________________ $\}\}$

$\{\{\, z \geq 0 \,\}\}$

Question 1b



$\{\{\, x < 3 \,\}\}$

`y = x + 4;`

$\{\{$ ________________________________ $\}\}$

`x = 2 * x;`

$\{\{$ ________________________________ $\}\}$

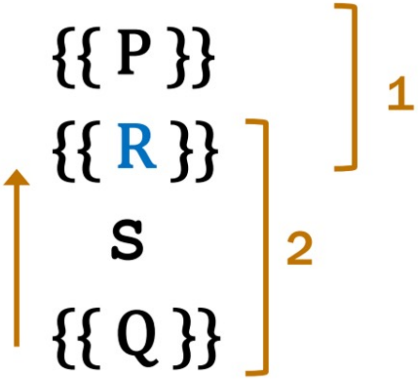
`y = y + x;`

$\{\{$ ________________________________ $\}\}$

$\{\{\, y < 14 \,\}\}$

Backward Reasoning – Review

- Backwards reasoning fills in preconditions
  - **Just use substitution!**
  - Gives weakest precondition making the triple valid
- Apply backwards reasoning to fill in R



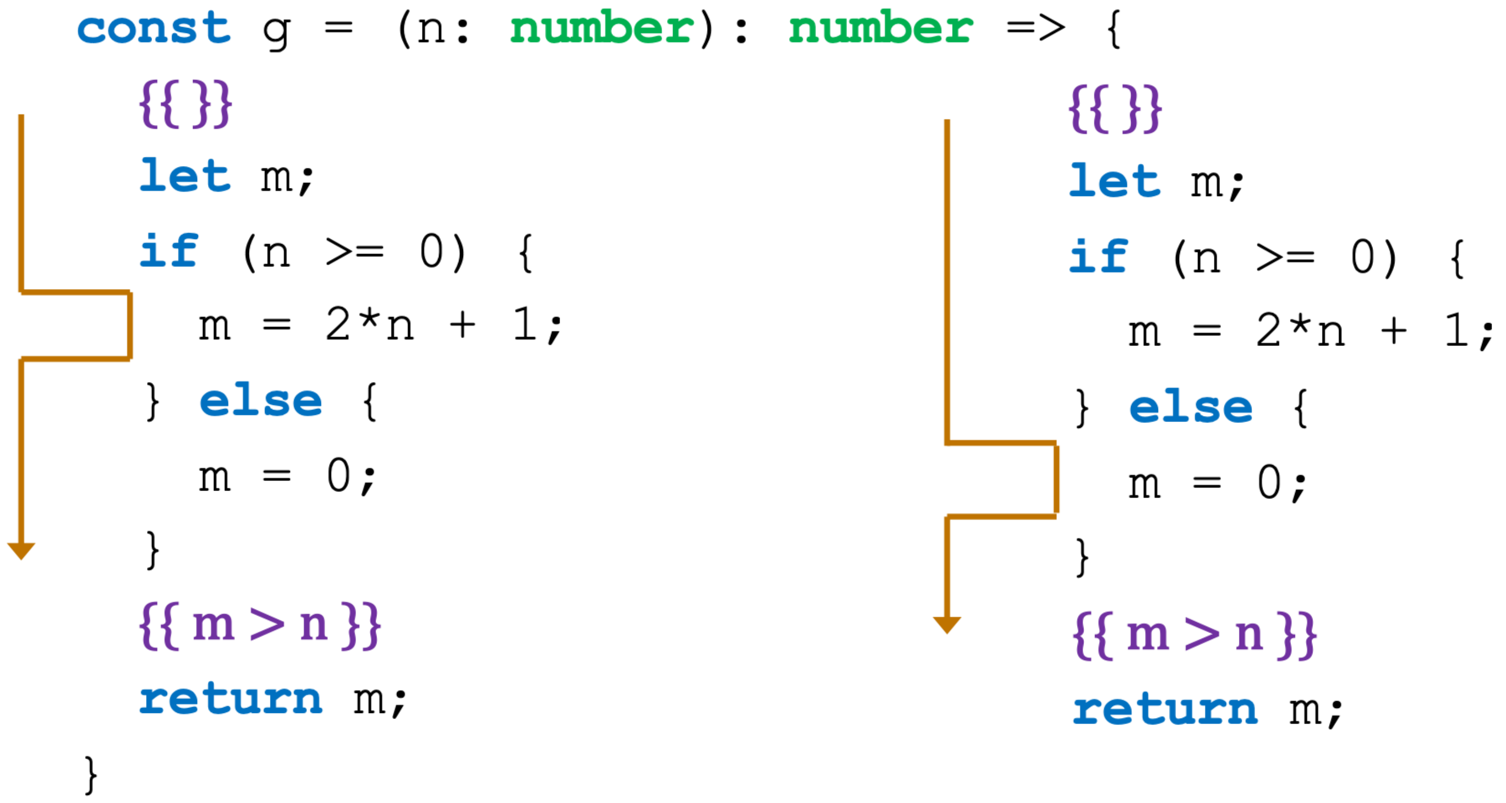
{{ P }}
{{ R }}
S
{{ Q }}

  - Check first triple by proving that P implies R

- **Good example problems in section worksheet!**

Conditionals – Review

- Reason through “then” and “else” branches independently
- Prove that each implies post condition

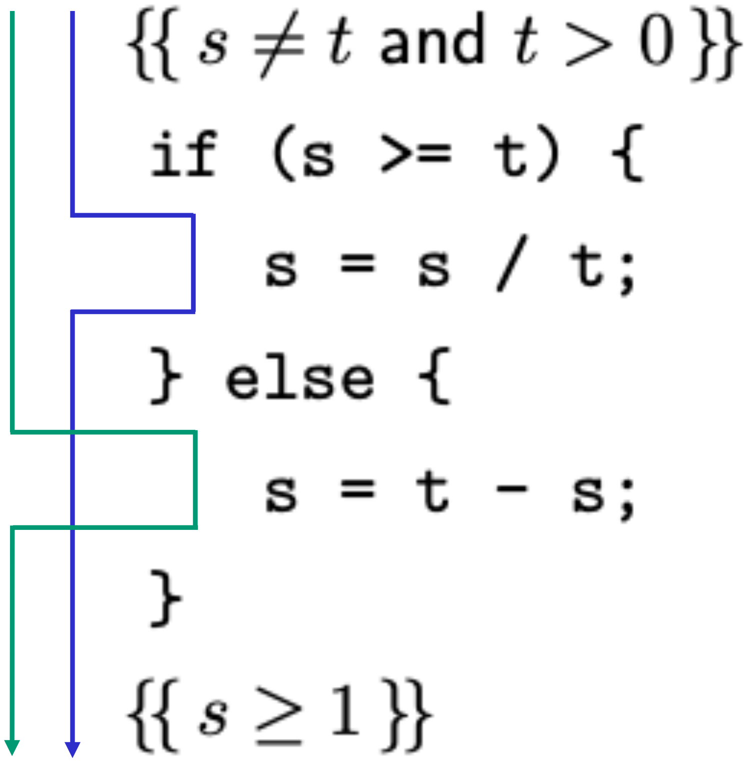


```
const g = (n: number): number => {
  {{ }}
  let m;
  if (n >= 0) {
    m = 2*n + 1;
  } else {
    m = 0;
  }
  {{ m > n }}
  return m;
}
```

```
  {{ }}
  let m;
  if (n >= 0) {
    m = 2*n + 1;
  } else {
    m = 0;
  }
  {{ m > n }}
  return m;
```

Question 3b

- Fill in the assertions for the “then” and “else” branches. Then complete two arguments showing that each postcondition implies $\{\{s \geq 1\}\}$

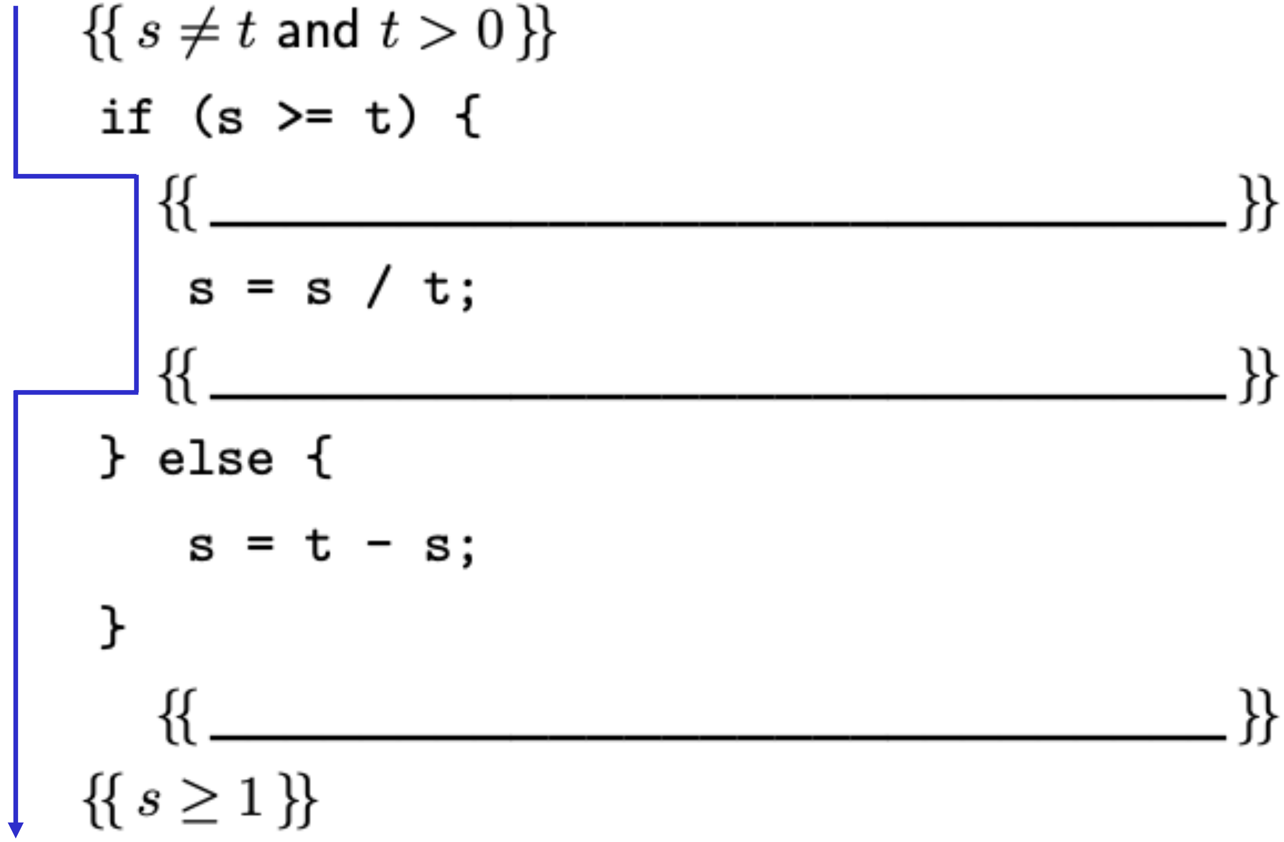


$\{\{ s \neq t \text{ and } t > 0 \}\}$

```
if (s >= t) {
    s = s / t;
} else {
    s = t - s;
}
```

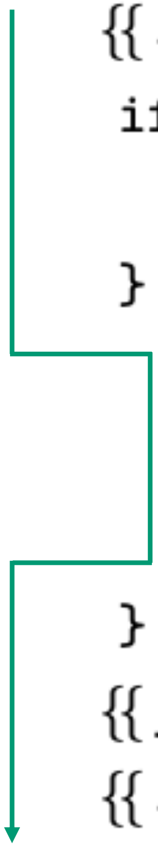
$\{\{ s \geq 1 \}\}$

Question 3b – “then” branch



```
$\{\{\, s \neq t \text{ and } t > 0 \,\}\}$
if (s >= t) {
    {{ ____________________________________ }}
    s = s / t;
    {{ ____________________________________ }}
} else {
    s = t - s;
}
    {{ ____________________________________ }}
$\{\{\, s \geq 1 \,\}\}$
```

Question 3b – “else” branch



$\{\{\, s \neq t \text{ and } t > 0 \,\}\}$

```
if (s >= t) {
    s = s / t;
} else {
```

$\{\{\,\rule{12cm}{0.4pt}\,\}\}$

```
    s = t - s;
```

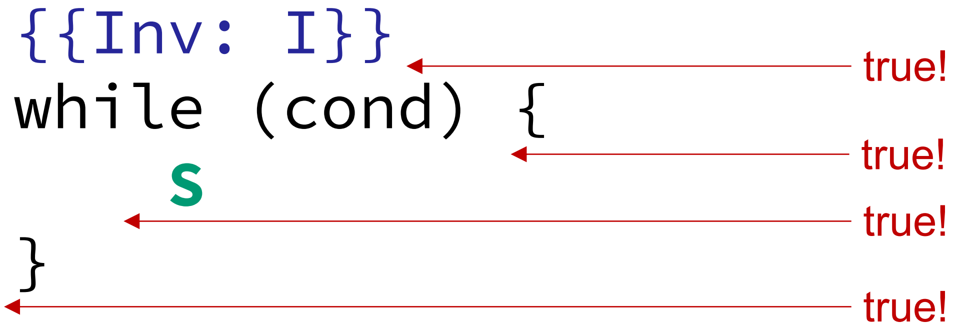
$\{\{\,\rule{12cm}{0.4pt}\,\}\}$

```
}
```

$\{\{\,\rule{12cm}{0.4pt}\,\}\}$

$\{\{\, s \geq 1 \,\}\}$

Loop Invariant – Review



```
{{Inv: I}}
while (cond) {
    S
}
```

true!

true!

true!

true!

- Loop invariant must be true **every time** at the top of the loop
  - The first time (before any iterations) and for the beginning of each iteration
- Also true every time at the bottom of the loop
  - Meaning it's true immediately after the loop exits
- During the body of the loop (during **S**), it isn't true

- Must use "`Inv`" notation to indicate that it's not a standard assertion

Well-Known Facts About Lists

- Feel free to cite these in your proofs! They're easily proven by structural induction (and you don't have to do that again)

- Lemma 2: **concat(L, nil) = L for any list L**

- Lemma 3: **rev(rev(L)) = L for any list L**

- **Lemma 4: concat(concat(L, R), S)**
  = concat(L, concat(R, S)) for any lists L, R, S

Question 4

Prove that the following code correctly calculates $\text{sum} - \text{abs}(L)$

(a) Invariant is true at top of loop the first time

```
let s: number = 0;
```

$$\{\{\ \text{Inv: } s + \text{sum-abs}(L) = \text{sum-abs}(L_0)\ \}\}$$

(c) Invariant is preserved by loop body

```
while (L !== nil) {
    if (L.hd < 0) {
        s = s + -L.hd;
    } else {
        s = s + L.hd;
    }
    L = L.tl;
}
```

(b) Postcondition holds when we exit

$$\{\{\ s = \text{sum-abs}(L_0)\ \}\}$$

Question 4a

Prove that the invariant is true at top of loop the first time

(a)

```
let s: number = 0;
{{ Inv: $s + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ }}
while (L !== nil) {
    if (L.hd < 0) {
        s = s + -L.hd;
    } else {
        s = s + L.hd;
    }
    L = L.tl;
}
{{ $s = \text{sum-abs}(L_0)$ }}
```

Question 4b

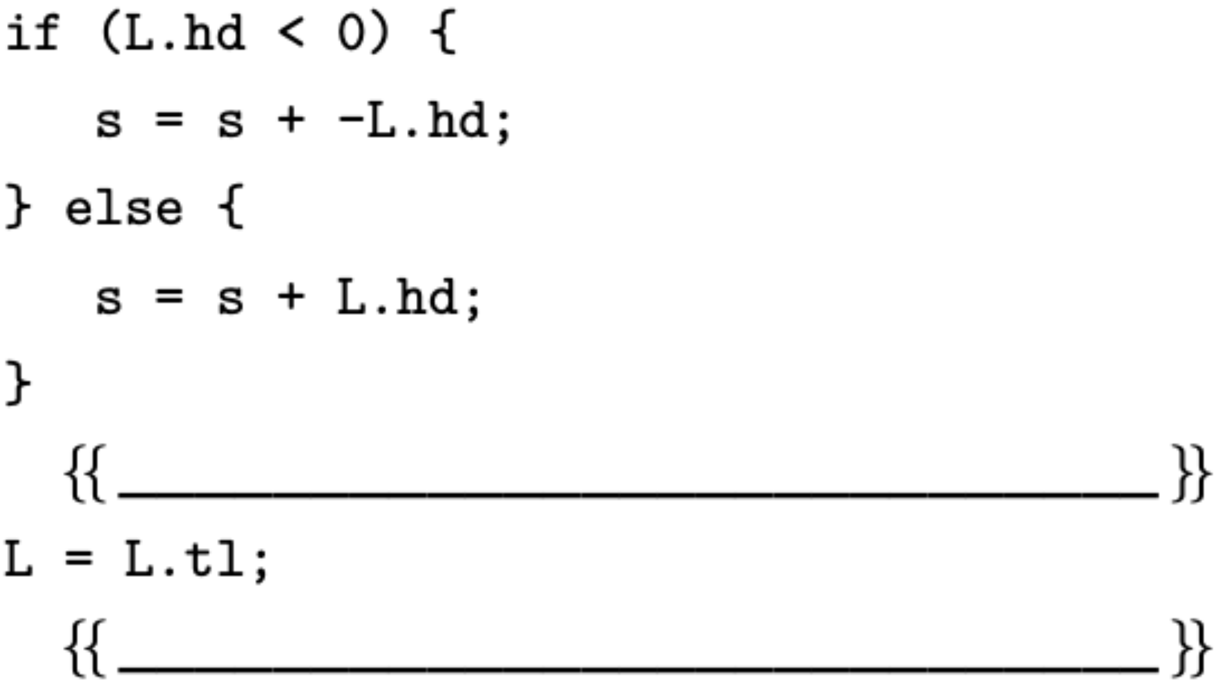
Prove that, when we exit the loop, the postcondition holds

```
let s: number = 0;
{{ Inv: $s + \text{sum-abs}(L) = \text{sum-abs}(L_0)$ }}
while (L !== nil) {
  if (L.hd < 0) {
    s = s + -L.hd;
  } else {
    s = s + L.hd;
  }
  L = L.tl;
}
(b) {{ $s = \text{sum-abs}(L_0)$ }}
```

Question 4c

Prove that the invariant is preserved by the body of the loop

(c)



```
if (L.hd < 0) {
    s = s + -L.hd;
} else {
    s = s + L.hd;
}
  {{________________________________}}
L = L.tl;
  {{________________________________}}
```

Question 4c

Prove that the invariant is preserved by the body of the loop

- Then, forward reasoning through the “then” branch

```
{{____________________________________________}}
if (L.hd < 0) {
  {{____________________________________________}}
   s = s + -L.hd;
  {{____________________________________________}}
} else {
  {{____________________________________________}}
   s = s + L.hd;
  {{____________________________________________}}
}
{{____________________________________________}}
L = L.tl;
{{____________________________________________}}
```

Question 4c

- Then, forward reasoning through the “else” branch

```
{{ ________________________________________ }}
if (L.hd < 0) {
  {{ ______________________________________ }}
   s = s + -L.hd;
  {{ ______________________________________ }}
} else {
  {{ ______________________________________ }}
   s = s + L.hd;
  {{ ______________________________________ }}
}
{{ ________________________________________ }}
L = L.tl;
{{ ________________________________________ }}
```

Question 4c

Then check that the ”then” branch implies the post condition:

Question 4c

Then check that the ”else” branch implies the post condition:

Question 6

(a) Give the invariant for the loop, based on the ”bottom-up” template for lists

(b) How do we initialize the variables so the invariant is true initially?

Question 6

(c) When do we exit the loop? What should the condition of the `while` be?

(d) Generally, the template says we move down the list with L = L.tl. swap processes 2 elements of the list at at time, so our loop should do the same. Write the loop body that does this and maintains the invariant: