CSE 331
Software Design & Implementation

Summer 2024

Section Weave – Imperative Programming I

Administrivia

- HW Weave released Thursday evening, due Wednesday at **11pm**
- Can submit as many times as you'd like until the deadline.
  - Use the autograder as a tool if you're not sure if your code/tests have bugs
- We will always tell you when you **can** use subscripts in forward/backwards reasoning: default rule (when not specified) is to **not** use subscripts

Hoare Triples – Review

- A Hoare Triple has 2 assertions and some code

  {{ P }}

  S

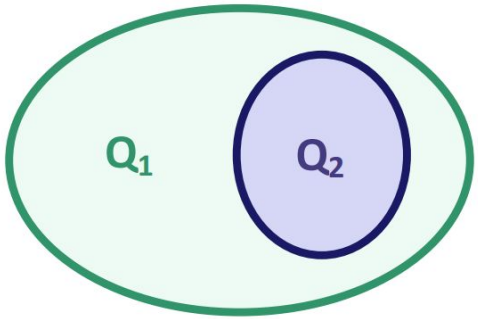
  {{ Q }}

  - 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, $Q_2 \rightarrow Q_1$

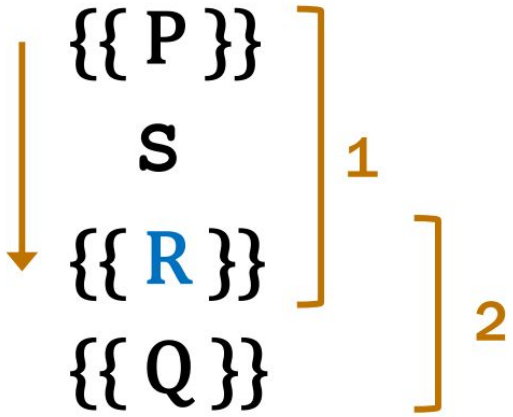


- Different from strength in *specifications*:
  - A stronger spec:
    - Stronger postcondition: guarantees more specific output
    - Weaker precondition: handles more allowable inputs

  compared to a weaker one

Forward Reasoning – Review

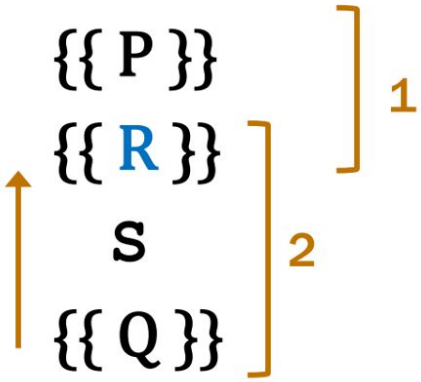
- Forwards reasoning fills in the postcondition
  - Gives strongest postcondition making the triple valid
- Apply forward reasoning to fill in R



  - Check second triple by proving that R implies Q

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



  - Check first triple by proving that P implies R

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

Question 1a

$\{\{ x \geq 4 \}\}$

```
y = x - 2n;
```

{{ _____ }}

```
z = 2n * y;
```

{{ _____ }}

```
z = z - 2n;
```

{{ _____ }}

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

Question 1b

$\{\{ x < 4 \}\}$

```
y = x + 4n;
```

{{ _____ }}

```
x = 2n * x;
```

{{ _____ }}

```
y = y + x;
```

{{ _____ }}

$\{\{ y < 16 \}\}$

Question 2a

$\{\{\, x < w + 1 \,\}\}$

$\{\{$ _____ $\}\}$

```
y = 3n * w;
```

$\{\{$ _____ $\}\}$

```
x = x * 3n;
```

$\{\{$ _____ $\}\}$

```
z = x - 9n;
```

$\{\{\, z < y \,\}\}$

Question 2b

$\{\{ x > 1 \}\}$

$\{\{$ _____ $\}\}$

```
y = x - 4n;
```

$\{\{$ _____ $\}\}$

```
z = 3n * y;
```

$\{\{$ _____ $\}\}$

```
z = z + 6n;
```

$\{\{ z \geq y \}\}$

Conditionals – Review

- Reason through “then” and “else” branches independently and combine last assertion of both branches with an “or” at the end
- Prove that each implies post condition by cases

```
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 3a

(a) Use forward reasoning to fill in the assertions. Then, combine the branches to assert the invariant we know at the end of the conditional and complete an argument by cases that this invariant implies $\{\{y \geq 2\}\}$.

Assume that x and y are both integers.

$\{\{x \geq 0\}\}$

```
if (x >= 6n) {
```

{{ _____ }}

```
  y = 2n * x - 10n;
```

{{ _____ }}

```
} else {
```

{{ _____ }}

```
  y = 20n - 3n * x;
```

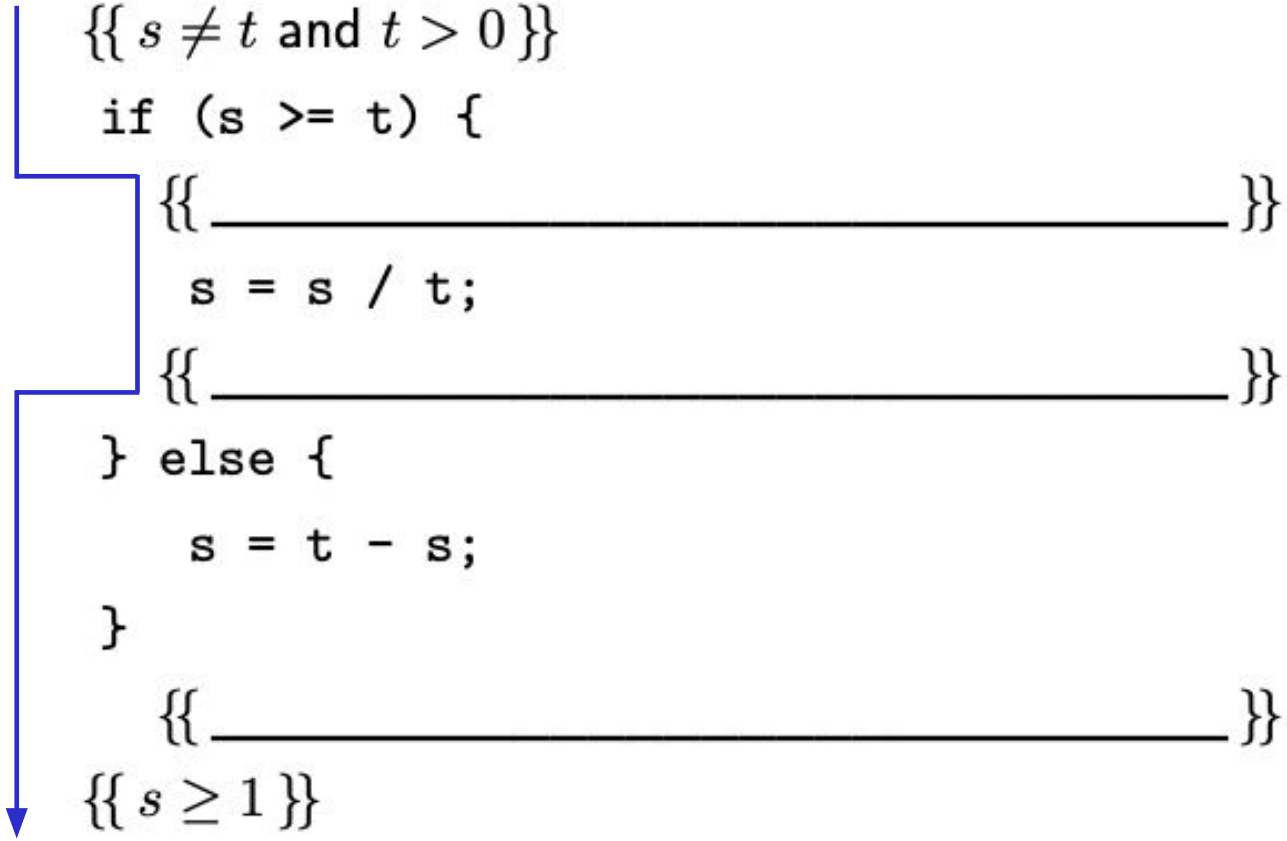
{{ _____ }}

```
}
```

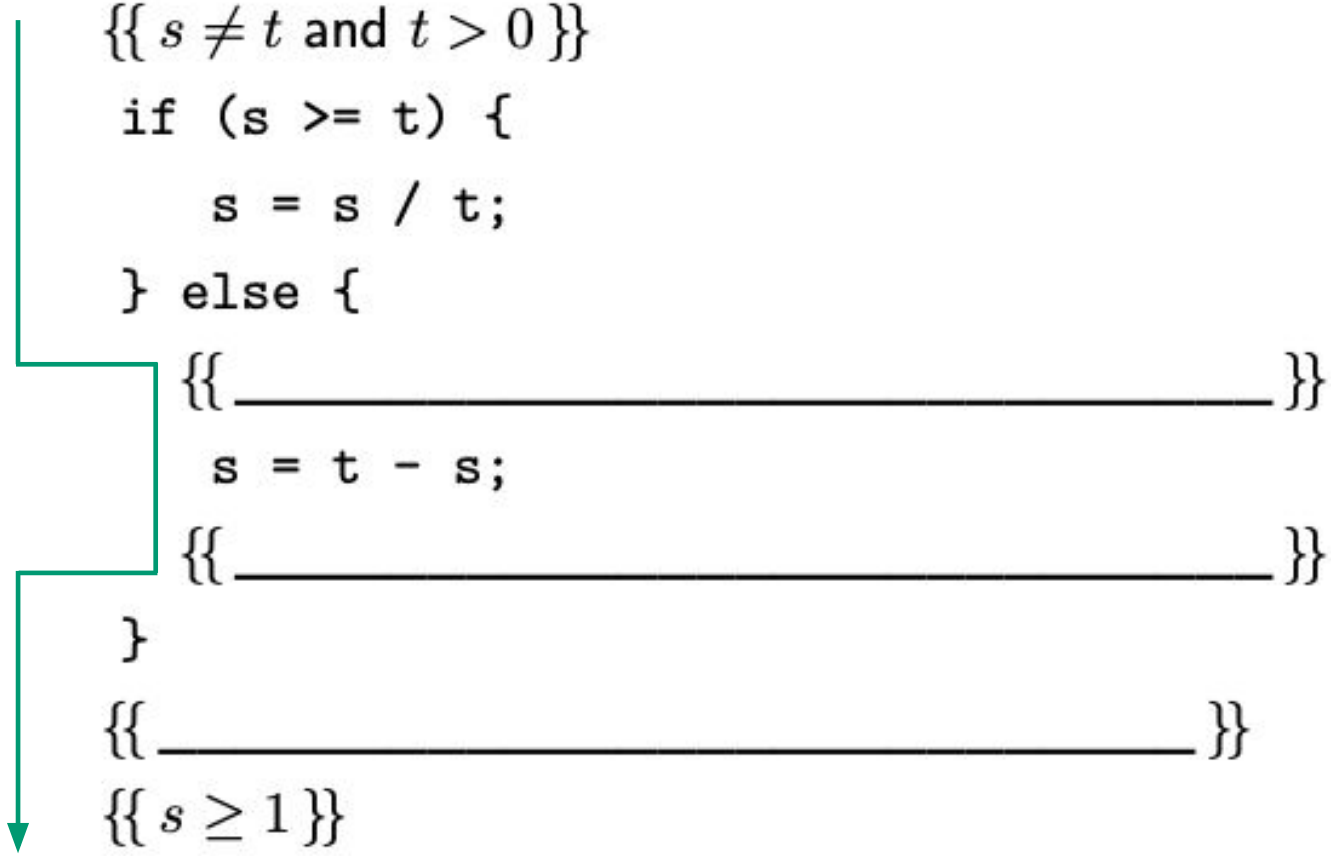
{{ _____ or _____ }}

$\{\{y \geq 2\}\}$

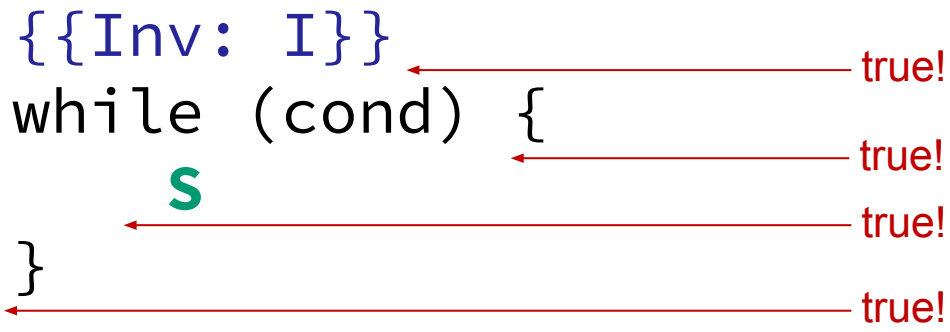
Question 3b – “then” branch



Question 3b – “else” branch



Loop Invariant – Review



- 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

(c) Invariant is preserved by loop body

(b) Postcondition holds when we exit

```
let s: bigint = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L.kind !== "nil") {
    if (L.hd < 0) {
        s = s + -L.hd;
    } else {
        s = s + L.hd;
    }
    L = L.tl;
}
{{ s = sum-abs(L_0) }}
```

Question 4a

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

(a)

```
let s: bigint = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L.kind !== "nil") {
    if (L.hd < 0) {
        s = s + -L.hd;
    } else {
        s = s + L.hd;
    }
    L = L.tl;
}
{{ s = sum-abs(L_0) }}
```

Question 4b

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

```
let s: bigint = 0;
{{ Inv: s + sum-abs(L) = sum-abs(L_0) }}
while (L.kind !== "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

- First, use backward reasoning to reason through last assignment

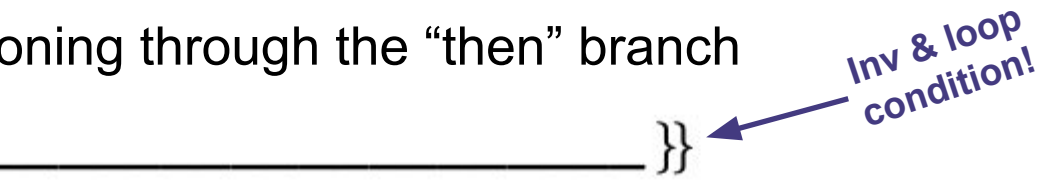
(c)

```
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;
}
```

{{ $s + \text{sum-abs}(L.\text{tl}) = \text{sum-abs}(L_0)$ }}

```
L = L.tl;
```

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

Question 4c

- Then, forward reasoning through the “else” branch

$\{\{\ s + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil}\ \}\}$

```
if (L.hd < 0) {
```

$\{\{\ s + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil and } L.\text{hd} < 0\ \}\}$

```
  s = s + -L.hd;
```

$\{\{\ s + L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil and } L.\text{hd} < 0\ \}\}$

```
} else {
```

$\{\{$ _____ $\}\}$

```
  s = s + L.hd;
```

$\{\{$ _____ $\}\}$

```
}
```

$\{\{\ s + \text{sum-abs}(L.\text{tl}) = \text{sum-abs}(L_0)\ \}\}$

```
L = L.tl;
```

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

Question 4c

$\{\{\ s + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil}\ \}\}$

```
if (L.hd < 0) {
```

$\{\{\ s + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil} \text{ and } L.\text{hd} < 0\ \}\}$

```
  s = s + -L.hd;
```

$\{\{\ s + L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil} \text{ and } L.\text{hd} < 0\ \}\}$

```
} else {
```

$\{\{\ s + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil} \text{ and } L.\text{hd} \geq 0\ \}\}$

```
  s = s + L.hd;
```

$\{\{\ s - L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil} \text{ and } L.\text{hd} \geq 0\ \}\}$

```
}
```

$\{\{\ (s + L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil} \text{ and } L.\text{hd} < 0)\ \}\}$

$\{\{\ \text{ or } (s - L.\text{hd} + \text{sum-abs}(L) = \text{sum-abs}(L_0) \text{ and } L \neq \text{nil} \text{ and } L.\text{hd} \geq 0)\ \}\}$

$\{\{\ s + \text{sum-abs}(L.\text{tl}) = \text{sum-abs}(L_0)\ \}\}$

```
L = L.tl;
```

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

Trees Review

invariant only for left-leaning trees

type Tree := empty
| node(val : $\mathbb{Z}$, left : Tree, right : Tree) with height(left) $\geq$ height(right)

func height(empty) := -1
height(node(x, S, T)) := $1 + \text{height}(S)$ for any $x : \mathbb{Z}$ and S, T : Tree

- height of nonempty tree is length of **longest** path to a leaf
- left-leaning tree: longest path is the one that travels towards the left child

Question 6

We can define the size of a tree, the number of values stored in it, as follows:

$$\begin{aligned} \textbf{func}\ \mathsf{size}(\mathsf{empty}) &:= 0 \\ \mathsf{size}(\mathsf{node}(x, S, T)) &:= 1 + \mathsf{size}(S) + \mathsf{size}(T) \quad \text{for any } x : \mathbb{Z} \text{ and } S, T : \mathsf{Tree} \end{aligned}$$

Prove by structural induction that, for any left-leaning tree U, we have

$$\mathsf{size}(U) \leq 2^{\mathsf{height}(U)+1} - 1$$