# CSE 331
# Software Design & Implementation

Winter 2026

Section 4 – Floyd Logic

# Administrivia

- HW 4 released tonight, due **Wednesday 2/4 at 11:59pm**

# Proof By Calculation – Review

- The goal of proof by calculation is to *show* that an assertion is true *given* facts that you already know
- You should ***start*** the proof with the left side of the assertion and ***end*** the proof with the right side of the assertion. Each symbol (=, >, <, etc.) connecting each line of the proof is that line's relationship to the **previous line on the proof**
- Only modify one side. Never do work on both sides. We can only work with what you have from the previous line, using definitions and facts.

# Structural Induction – Review

- Let **P(S)** be the claim
- To Prove P(S) holds for any list S, we need to prove two implications: base case and inductive case
  - **Base Case:** prove P(nil)
    - Use any known facts and definitions
  - **Inductive Hypothesis:** assume **P(L)** is true for a L: List
    - Use this in the inductive step ONLY ⤵️
  - **Inductive Step:** prove **P(x :: L)** for any x : *Z,* L : List
    - Direct proof
    - Use known facts and definitions and Inductive Hypothesis
- Assuming we know P(L), if we prove P(x :: L), we then prove recursively that P(S) holds for any List

# Hoare Triples – Review

- A Hoare Triple has 2 assertions and some code

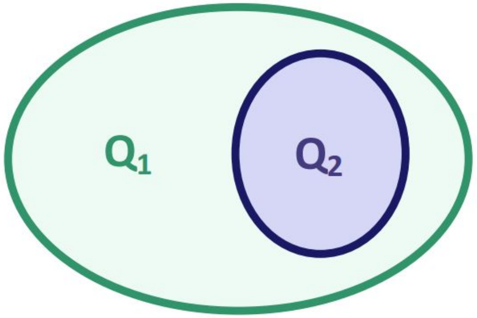
  {{ P }}
  S
  {{ Q }}

  - 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
    - We use Proof By Calculation to prove our Hoare Triples!

# 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*

# Question …

Which is the strongest assertion:

- $x > 3$
- $x \geq 3$
- $x > 3$ and $x \in \{2, 4, 6, 8, 10\}$
- $x > 3$ and $x \% 2 = 0$

Discuss with the person next to you

# Question …

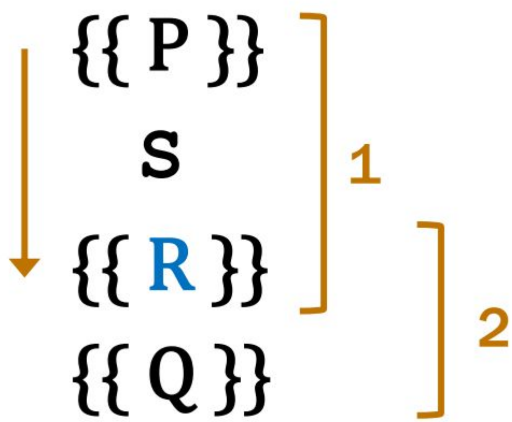
Which is the strongest assertion:

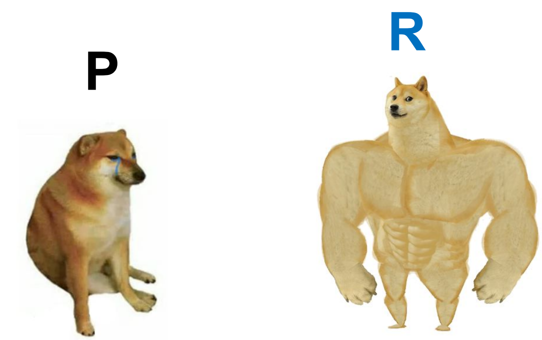
- $x > 3$
- $x \geq 3$
- **$x > 3$ and $x \in \{2, 4, 6, 8, 10\}$**
- $x > 3$ and $x \% 2 = 0$

Discuss with the person next to you

# 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

# Forward Reasoning Error Example

```
{{ x > 1 }}
x = x + 1;
{{ x = x₀ + 1 and x₀ > 1 }}
y = 3 * x;
{{ x = x₀ + 1 and y = 3 * x }}
z = y + 1;
{{ x = x₀ + 1 and z = (3 * x) + 1 }}
```

Drops this assertion

What's wrong with these assertions?

Uses subscripts for an invertible operation

Simplifies assertions too early by dropping y variable relationship to x

# Corrected Forward Reasoning Example

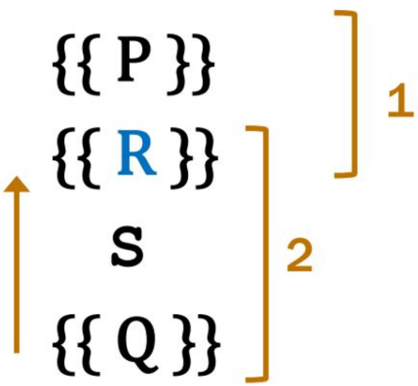
```
{{ x > 1 }}
x = x + 1;
{{ x - 1 > 1 }}
y = 3 * x;
{{ x - 1 > 1 and y = 3 * x }}
z = y + 1
{{ x - 1 > 1 and y = 3 * x and z = y + 1 }}
```

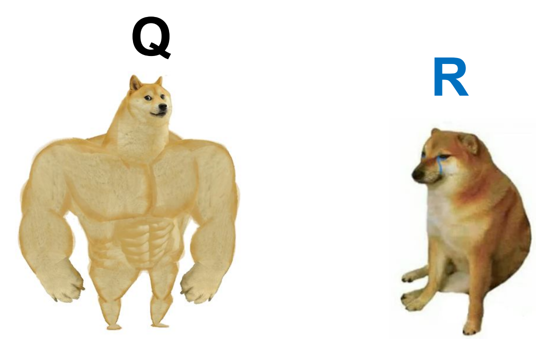
does not simplify assertions early or drop variable relationships

updates x for this operation rather than introducing subscripts

# 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

# Forward & Backward General Rules

## Forward Reasoning:

- After each line of code *update* variables in assertions based on how they they were changed by the line of code

## Backward Reasoning:

- As you work your way up the code *directly* substitute how variables are modified in the code into your assertions

## General:

- Do **not** drop or simplify assertions
- Do **not** use subscripts for invertible operations (addition and subtraction are *always* invertible)

# 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
- **Note**: this is important for your homework!

```
public static int g(int n) {
    {{ }}
    int m = 0;
    if (n >= 0) {
        m = 2 * n + 1;
    } else {
        m = 0;
    }
    {{m > n}}
    return m;
}
```

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

# Loop Invariant – Review

```
{{Inv: I}}
                      ← true!
while (cond) {
                      ← true!
    S
                      ← true!
}
                      ← true!
```

- Loop invariant must be true **<u>every time</u>** 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

# Question ….

Where is it allowed for a loop invariant not to hold?

- before the loop
- after the loop
- after entering the loop
- before exiting the loop
- during the code execution inside of the loop

# Question ….

Where is it allowed for a loop invariant not to hold?

- before the loop
- after the loop
- after entering the loop
- before exiting the loop
- **during the code execution inside of the loop**