Review: Straight-line Code
Forward & Backward Reasoning

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} \\
S \\
\{\{ ? \}\}
\end{align*}
\]

- P is what we know initially
- Work downward
- Determine what holds after S executes

Backward reasoning

\[
\begin{align*}
\{\{ ? \}\} \\
S \\
\{\{ Q \}\}
\end{align*}
\]

- Q is what we want at the end
- Work upward
- Determine what must hold initially before S executes
Assignment Rule

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} \\
x &= \text{expr} \\
\{\{ ? \}\}
\end{align*}
\]
Assignment Rule

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} \\
x &= \text{expr} \\
\{\{ P \text{ and } x = \text{expr} \}\}
\end{align*}
\]

• adds another known fact
• these tend to accumulate…
  – many are irrelevant

(above assumes \(x\) not used in \(P\))
Assignment Rule

Forward reasoning

\[
\{\{ P \}\}\n\]
\[
x = \text{expr};
\]
\[
\{\{ P \text{ and } x = \text{expr} \}\}\n\]

• adds another known fact
• these tend to accumulate…
  – many are irrelevant

(above assumes \(x\) not used in \(P\))

Backward reasoning

\[
\{\{ ? \}\}\n\]
\[
x = \text{expr};
\]
\[
\{\{ Q \}\}\n\]
Assignment Rule

Forward reasoning

\{\{ P \}\}
\[
x = expr;
\]
\{\{ P \land x = expr \}\}

• adds another known fact
• these tend to accumulate…
  – many are irrelevant

(above assumes \( x \) not used in P)

Backward reasoning

\{\{ Q[x=expr] \}\}
\[
x = expr;
\]
\{\{ Q \}\}

• just substitution
• most general conditions for getting Q after \( x = expr; \)
Assignment Example

Forward reasoning

\[
\begin{align*}
\{ & w = 3 \} \\
& x = y - 5; \\
& \{ & ? \} \\
\end{align*}
\]
Assignment Example

Forward reasoning

\[
\{ w = 3 \}
\]
\[
x = y - 5;
\]
\[
\{ w = 3 \text{ and } x = y - 5 \}\]
Assignment Example

Forward reasoning

$$\{ \{ w = 3 \} \}$$
$$x = y - 5;$$
$$\{ \{ w = 3 \text{ and } x = y - 5 \} \}$$

Backward reasoning

$$\{ \{ ? \} \}$$
$$x = y - 5;$$
$$\{ \{ w = x + 5 \} \}$$
Assignment Example

Forward reasoning

\[
\begin{align*}
\{\ w = 3 \ \} \\
x &= y - 5; \\
\{\ w = 3 \text{ and } x = y - 5 \ \}
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{\ w = y \ \} \\
x &= y - 5; \\
\{\ w = x + 5 \ \}
\end{align*}
\]
Sequence Rule

Forward reasoning

{{ P }}
S1
S2
{{ ? }}
Sequence Rule

Forward reasoning

{{ \text{P} \}}
S1
{{ \text{?} \}}
S2
{{ \text{?} \}}
Sequence Rule

Forward reasoning

\[
\begin{align*}
\{ \{ P \} \} \\
S1 \\
\{ \{ P1 \} \} \\
S2 \\
\{ \{ ? \} \}
\end{align*}
\]
Sequence Rule

Forward reasoning

{{ P }}
S1

{{ P1 }}
S2

{{ P2 }}
### Sequence Rule

<table>
<thead>
<tr>
<th>Forward reasoning</th>
<th>Backward reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{ P }}</td>
<td>{{ ? }}</td>
</tr>
<tr>
<td>S1</td>
<td>S1</td>
</tr>
<tr>
<td>{{ P1 }}</td>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
<td>{{ Q }}</td>
</tr>
<tr>
<td>{{ P2 }}</td>
<td></td>
</tr>
</tbody>
</table>
## Sequence Rule

<table>
<thead>
<tr>
<th>Forward reasoning</th>
<th>Backward reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{ P }} S1</td>
<td>{{ ? }} S1</td>
</tr>
<tr>
<td>{{ P1 }} S2</td>
<td>{{ ? }} S2</td>
</tr>
<tr>
<td>{{ P2 }}</td>
<td>{{ Q }}</td>
</tr>
</tbody>
</table>
Sequence Rule

Forward reasoning

\[
\begin{align*}
&\{P\} \\
&S1 \\
&\{P1\} \\
&S2 \\
&\{P2\}
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
&\{?\} \\
&S1 \\
&\{Q2\} \\
&S2 \\
&\{Q\}
\end{align*}
\]
Sequence Rule

Forward reasoning

```
{{ P }}
S1
{{ P1 }}
S2
{{ P2 }}
```

Backward reasoning

```
{{ Q1 }}
S1
{{ Q2 }}
S2
{{ Q }}
```
If-Statement Rule

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} \\
\text{if (cond)} \\
\quad S1 \\
\text{else} \\
\quad S2 \\
\{\{ \text{?} \}\}
\end{align*}
\]
If-Statement Rule

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} \\
\text{if (cond)} \Rightarrow \{\{ P \text{ and cond} \}\} \\
S1 \quad \text{else} \Rightarrow \{\{ P \text{ and not cond} \}\} \\
S2 \Rightarrow \{\? \}\n\end{align*}
\]
If-Statement Rule

Forward reasoning

```
{{ P }}
if (cond)
  {{ P and cond }}
  S1
  {{ P1 }}
else
  {{ P and not cond }}
  S2
  {{ P2 }}
{{ ? }}
```
If-Statement Rule

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} \\
\text{if (cond)} \\
\{\{ P \text{ and } cond \}\} \\
S1 \\
\{\{ P1 \}\} \\
\text{else} \\
\{\{ P \text{ and } not \text{cond} \}\} \\
S2 \\
\{\{ P2 \}\} \\
\{\{ P1 \text{ or } P2 \}\}
\end{align*}
\]
If-Statement Rule

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} \\
\text{if (cond)} \\
\{\{ P \text{ and } \text{cond} \}\} \\
S1 \\
\{\{ P1 \}\} \\
\text{else} \\
\{\{ P \text{ and not } \text{cond} \}\} \\
S2 \\
\{\{ P2 \}\} \\
\{\{ P1 \text{ or } P2 \}\}
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{\{ ? \}\} \\
\text{if (cond)} \\
S1 \\
\text{else} \\
S2 \\
\{\{ Q \}\}
\end{align*}
\]
If-Statement Rule

Forward reasoning

\[
\begin{align*}
\{\{ P \}\} & \quad \text{if (cond)} \\
& \quad \{\{ P \text{ and } \text{cond} \}\} \\
& \quad S1 \\
& \quad \{\{ P1 \}\} \\
\text{else} & \quad \{\{ P \text{ and not } \text{cond} \}\} \\
& \quad S2 \\
& \quad \{\{ P2 \}\} \\
& \quad \{\{ P1 \text{ or } P2 \}\}
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{\{ ? \}\} & \quad \text{if (cond)} \\
& \quad S1 \\
& \quad \{\{ Q \}\} \\
\text{else} & \quad \{\{ Q \}\} \\
& \quad \{\{ Q \}\}
\end{align*}
\]
If-Statement Rule

Forward reasoning

\[
\{\{ P \}\} \\
\text{if (cond)} \\
\{\{ P \text{ and } \text{cond} \}\} \\
S1 \\
\{\{ P1 \}\} \\
\text{else} \\
\{\{ P \text{ and not } \text{cond} \}\} \\
S2 \\
\{\{ P2 \}\} \\
\{\{ P1 \text{ or } P2 \}\}
\]

Backward reasoning

\[
\{\{ ? \}\} \\
\text{if (cond)} \\
\{\{ Q1 \}\} \\
S1 \\
\{\{ Q \}\} \\
\text{else} \\
\{\{ Q2 \}\} \\
S2 \\
\{\{ Q \}\} \\
\{\{ Q \}\}
\]
If-Statement Rule

Forward reasoning

\{
\{P\} \\
if (cond) \\
\{P \text{ and } \text{cond}\} \\
S1 \\
\{P1\} \\
else \\
\{P \text{ and not } \text{cond}\} \\
S2 \\
\{P2\} \\
\{P1 \text{ or } P2\}
\}

Backward reasoning

\{
\{Q1 \text{ and } \text{cond or} \ \\
Q2 \text{ and not } \text{cond}\} \\
if (cond) \\
\{Q1\} \\
S1 \\
\{Q\} \\
else \\
\{Q2\} \\
S2 \\
\{Q\} \\
\{Q\}
\}
If-Statement Example

Forward reasoning

```c
{{}}
if (x >= 0)
    y = x;
else
    y = -x;
{{ ? }}
```
If-Statement Example

Forward reasoning

```plaintext
{{ }}
if (x >= 0)
  {{ x >= 0 }}
  y = x;
else
  {{ x < 0 }}
  y = -x;
{{ ? }}
```
Forward reasoning

```plaintext
{{ }}
if (x >= 0)
    {{ x >= 0 }}
    y = x;
    {{ x >= 0 and y = x }}
else
    {{ x < 0 }}
    y = -x;
    {{ x < 0 and y = -x }}
{{ ? }}
```
If-Statement Example

Forward reasoning

```plaintext
{}
if \( x \geq 0 \)
{} \( x \geq 0 \)
y = x;
{} \( x \geq 0 \) and \( y = x \)
else
{} \( x < 0 \)
y = -x;
{} \( x < 0 \) and \( y = -x \)
{} \((x \geq 0 \) and \( y = x \)) or \((x < 0 \) and \( y = -x \))
```

CSE 331 Spring 2016
If-Statement Example

Forward reasoning

{\{\} }
if (x >= 0)
{\{ x >= 0 \} }
y = x;
{\{ x >= 0 and y = x \} }
else
{\{ x < 0 \} }
y = -x;
{\{ x < 0 and y = -x \} }
{\{ y = |x| \} }
If-Statement Example

Forward reasoning

```plaintext
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = x;
  {{ x >= 0 and y = x }}
else
  {{ x < 0 }}
y = -x;
  {{ x < 0 and y = -x }}
{{ y = |x| }}
```

Backward reasoning

```plaintext
{{ ? }}
if (x >= 0)
  {{ x >= 0 }}
y = x;
else
  {{ x < 0 }}
y = -x;
{{ y = |x| }}
```
If-Statement Example

Forward reasoning

```
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = x;
  {{ x >= 0 and y = x }}
else
  {{ x < 0 }}
y = -x;
  {{ x < 0 and y = -x }}
{{ y = |x| }}
```

Backward reasoning

```
{{ ? }}
if (x >= 0)
  y = x;
  {{ y = |x| }}
else
  y = -x;
  {{ y = |x| }}
  {{ y = |x| }}
```
If-Statement Example

Forward reasoning

\[
\begin{align*}
\{ \} \\
\text{if } (x >= 0) \\
\{ x >= 0 \} \\
y = x; \\
\{ x >= 0 \text{ and } y = x \} \\
\text{else} \\
\{ x < 0 \} \\
y = -x; \\
\{ x < 0 \text{ and } y = -x \} \\
\{ y = |x| \}
\end{align*}
\]

Backward reasoning

\[
\begin{align*}
\{ ? \} \\
\text{if } (x >= 0) \\
\{ x = |x| \} \\
y = x; \\
\{ y = |x| \} \\
\text{else} \\
\{ -x = |x| \} \\
y = -x; \\
\{ y = |x| \} \\
\{ y = |x| \}
\end{align*}
\]
If-Statement Example

Forward reasoning

```c
{{ }}
if (x >= 0)
    {{ x >= 0 }}
y = x;
    {{ x >= 0 and y = x }}
else
    {{ x < 0 }}
y = -x;
    {{ x < 0 and y = -x }}
{{ y = |x| }}
```

Backward reasoning

```c
{{ ? }}
if (x >= 0)
    {{ x >= 0 }}
y = x;
    {{ y = |x| }}
else
    {{ x <= 0 }}
y = -x;
    {{ y = |x| }}
{{ y = |x| }}
```
If-Statement Example

Forward reasoning

{{ }}
if (x >= 0)
    {{ x >= 0 }}
y = x;
    {{ x >= 0 and y = x }}
else
    {{ x < 0 }}
y = -x;
    {{ x < 0 and y = -x }}
{{ y = |x| }}

Backward reasoning

{{ (x >= 0 and x >= 0) or (x < 0 and x <= 0) }}
if (x >= 0)
    {{ x >= 0 }}
y = x;
    {{ y = |x| }}
else
    {{ x <= 0 }}
y = -x;
    {{ y = |x| }}
{{ y = |x| }}
If-Statement Example

Forward reasoning

```plaintext
{{
if (x >= 0)
  {{ x >= 0 }}</x >= 0>
  y = x;
  {{ x >= 0 and y = x }}</x >= 0>
else
  {{ x < 0 }}</x < 0>
  y = -x;
  {{ x < 0 and y = -x }}</x < 0>
{{ y = |x| }}</y = |x| >
```}

Backward reasoning

```plaintext
{{ x >= 0 or x < 0 }
if (x >= 0)
  {{ x >= 0 }}</x >= 0>
  y = x;
  {{ y = |x| }}</y = |x| >
else
  {{ x <= 0 }}</x <= 0>
  y = -x;
  {{ y = |x| }}</y = |x| >
{{ y = |x| }}</y = |x| >
```
If-Statement Example

Forward reasoning

```plaintext
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = x;
  {{ x >= 0 and y = x }}
else
  {{ x < 0 }}
y = -x;
  {{ x < 0 and y = -x }}
{{ y = |x| }}
```

Backward reasoning

```plaintext
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = x;
  {{ y = |x| }}
else
  {{ x <= 0 }}
y = -x;
  {{ y = |x| }}
{{ y = |x| }}
```
Verifying Correctness (*Inspection*)

Two different ways of checking \{\{ P \}\} \implies \{\{ Q \}\}

**Use forward reasoning:**

\[ \{\{ P \}\} \implies \{\{ Q' \}\} \]

- Find \( Q' \) assuming \( P \).
- Check that \( Q' \) implies \( Q \).
  - weaken postcondition

**Use backward reasoning:**

\[ \{\{ P' \}\} \implies \{\{ Q \}\} \]

- Find \( P' \) that produces \( Q \).
- Check that \( P \) implies \( P' \).
  - strengthen precondition

You know how to verify correctness of straight-line code.

You will do this on HW1.
Using Both Forward & Backward

Also possible to check correctness by mixing forward & backward:

```
if (x >= 0)
  y = div(x,2);
else
  y = -div(-x+1,2);
{ 2y = x or 2y = x - 1 }
```

Assume that \( \text{div}(a,b) \) computes \( a/b \) rounded toward zero.
Code to compute \( x/2 \) rounded toward minus infinity (usual division).
Using Both Forward & Backward

Also possible to check correctness by mixing forward & backward:

```c
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = div(x, 2);
else
  {{ x < 0 }}
y = -div(-x + 1, 2);
{{ 2y = x or 2y = x - 1 }}
```
Using Both Forward & Backward

Also possible to check correctness by mixing forward & backward:

```
if (x >= 0)
    { x >= 0 }
    y = div(x, 2);
    { 2y = x or 2y = x - 1 }
else
    { x < 0 }
    y = -div(-x+1, 2);
    { 2y = x or 2y = x - 1 }
```
Using Both Forward & Backward

Also possible to check correctness by mixing forward & backward:

```plaintext
{{ }}
if (x >= 0)
  {{ x >= 0 }}
y = div(x,2);
  {{ 2y = x or 2y = x - 1 }}
else
  {{ x < 0 }}
y = -div(-x+1,2);
  {{ 2y = x or 2y = x - 1 }}
{{ 2y = x or 2y = x - 1 }}
```
Using Both Forward & Backward

Also possible to check correctness by mixing forward & backward:

```c
if (x >= 0)
    y = div(x, 2);
    2y = x or 2y = x - 1
else
    y = -div(-x + 1, 2);
    2y = x or 2y = x - 1
```
Loops
Loop Invariant

A loop invariant is one that always holds at the top of the loop:

\[
\{
\text{Inv: } I
\}\\
\text{while (cond)}\\
S
\]

- It holds when we first get to the loop.
- It holds each time we execute \( S \) and come back to the top.

Notation: I’ll use “Inv:” to indicate a loop invariant.
While-Loop Rule

Consider a while-loop (other loop forms not too different):

\[
\begin{align*}
\{\{P\}\} & \quad \text{while } \ (\text{cond}) \quad S \quad \{\{Q\}\}
\end{align*}
\]

This triple is valid iff: there is a loop invariant \(I\) such that

\[
\begin{align*}
\{\{P\}\} \quad \{\{\text{Inv: } I\}\} & \quad \text{while } \ (\text{cond}) \quad S \quad \{\{Q\}\}
\end{align*}
\]

- \(I\) holds initially
- \(I\) holds each time we execute \(S\)
- \(Q\) holds when \(I\) holds and \(\text{cond}\) is false
While-Loop Rule

Consider a while-loop (other loop forms not so different):

\[
\{\{ P \}\} \text{ while (cond) } S \{\{ Q \}\}
\]

This triple is valid iff: there is a loop invariant I such that

\[
\{\{ P \}\} \\
\{\{ \text{Inv: I} \}\} \\
\text{while (cond) } S \\
\{\{ Q \}\}
\]

- P implies I
- I holds each time we execute S
- Q holds when I holds and cond is false
While-Loop Rule

Consider a while-loop (other loop forms not so different):

\[
\{\{ P \} \} \text{ while } (\text{cond}) \ S \ \{\{ Q \} \}
\]

This triple is valid iff: there is a loop invariant \( I \) such that

\[
\begin{align*}
\{\{ P \} \} \\
\{\{ \text{Inv: } I \} \} \\
\text{while } (\text{cond}) \\
S \\
\{\{ Q \} \}
\end{align*}
\]

- \( P \) implies \( I \)
- \( \{\{ I \text{ and cond} \} \} \ S \ \{\{ I \} \} \text{ is valid} \)
- \( Q \) holds when \( I \) holds and \( \text{cond} \) is false
While-Loop Rule

Consider a while-loop (other loop forms not so different):

\[
\{\{ P \}\} \text{ while (cond) } S \{\{ Q \}\}
\]

This triple is valid iff: there is a loop invariant I such that

\[
\begin{align*}
\{\{ P \}\} & \quad \text{P implies I} \\
\{\{ \text{Inv: I} \}\} & \quad \text{\{I and cond\} S \{I\} is valid} \\
\text{while (cond) } S & \quad \text{(I and not cond) implies Q} \\
\{\{ Q \}\} &
\end{align*}
\]
While-Loop Rule

Consider a while-loop (other loop forms not so different):

\[
\{\!\{ P \}!\} \text{ while (cond)} \ S \ \{\!\{ Q \}!\}
\]

This triple is valid iff: there is a loop invariant \( I \) such that

\[
\begin{align*}
\{\!\{ P \}!\} & \quad \cdot \quad \text{P implies I} \\
\{\!\{ \text{Inv: } I \}!\} & \quad \cdot \quad \{\!\{ I \text{ and cond} \}!\} \ S \ \{\!\{ I \}!\} \text{ is valid} \\
\text{while (cond)} \ S & \quad \cdot \quad (I \text{ and not cond}) \text{ implies Q} \\
\{\!\{ Q \}!\} & \\
\end{align*}
\]
More on Loop Invariants

- We need a loop invariant to check validity of a while loop.
- There is no automatic way to generate these.
  - (A theory course will explain why…)

- For this lecture, all loop invariants will be given.
- Next lecture will discuss how to choose a loop invariant.

- Pro Tip: should almost always document your loop invariants
  - as we just saw, much easier for others to check your code
  - possible exception for loops that are “obvious”
- Pro Tip: with a good loop invariant, the code is easy to write
Example: sum of array

Consider the following code to compute \( b[0] + \ldots + b[n-1] \):

\[
\begin{align*}
\{ & \{ \text{b.length} \geq n \} \\
    & \text{s} = 0; \\
    & \text{i} = 0; \\
    & \text{while} \ (i \neq n) \ \{ \\
    & \quad \text{s} = \text{s} + \text{b}[i]; \\
    & \quad \text{i} = \text{i} + 1; \\
    & \} \\
\{ & \{ \text{s} = b[0] + \ldots + b[n-1] \} \}
\end{align*}
\]
Example: sum of array

Consider the following code to compute $b[0] + ... + b[n-1]$:

```java
{{ b.length >= n }}
int s = 0;
int i = 0;
{{ Inv: s = b[0] + ... + b[i-1] }}
while (i != n) {
    s = s + b[i];
    i = i + 1;
}
{{ s = b[0] + ... + b[n-1] }}
```
Example: sum of array

Consider the following code to compute $b[0] + \ldots + b[n-1]$

```java
{{ b.length >= n }}
s = 0;
i = 0;
{{ s = 0 and i = 0 }}
{{ Inv: s = b[0] + \ldots + b[i-1] }}
while (i != n) {
    s = s + b[i];
i = i + 1;
}
{{ s = b[0] + \ldots + b[n-1] }}
```
Example: sum of array

Consider the following code to compute $b[0] + ... + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = 0;
{{ s = 0 and i = 0 }}
{{ Inv: s = b[0] + ... + b[i-1] }}
while (i != n) {
    s = s + b[i];
i = i + 1;
}
{{ s = b[0] + ... + b[n-1] }}
```
Example: sum of array

Consider the following code to compute $b[0] + \ldots + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = 0;
{{ s = 0 and i = 0 }}
{{ Inv: s = b[0] + \ldots + b[i-1] }}
while (i != n) {
    s = s + b[i];
i = i + 1;
}
{{ s = b[0] + \ldots + b[n-1] }}
```

• (s = 0 and i = 0) implies $s = b[0] + \ldots + b[i-1]$?

Yes. (An empty sum is zero.)
Example: sum of array

Consider the following code to compute $b[0] + \ldots + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = 0;
{{ s = 0 and i = 0 }}
{{ Inv: s = b[0] + \ldots + b[i-1] }}
while (i != n) {
    s = s + b[i];
    i = i + 1;
}
{{ s = b[0] + \ldots + b[n-1] }}
```

• $(s = 0$ and $i = 0)$ implies I
Example: sum of array

Consider the following code to compute \( b[0] + \ldots + b[n-1] \):

\[
\begin{align*}
\text{\{b.length} & \geq \text{n\}} \\
\text{s} & = 0; \\
\text{i} & = 0; \\
\text{\{Inv: s = b[0] + \ldots + b[i-1]\}} & \text{ implies } I \\
\text{while (i} & \neq \text{n) \{} \\
\text{\{s = b[0] + \ldots + b[i-1] and i} & \neq \text{n\}} & \\
\text{s} & = \text{s} + b[i]; \\
\text{i} & = \text{i} + 1; \\
\text{\{s = b[0] + \ldots + b[i-1]\}} & \\
\text{\}} \\
\text{\{s = b[0] + \ldots + b[n-1]\}}
\end{align*}
\]
Example: sum of array

Consider the following code to compute $b[0] + \ldots + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = 0;
{{ Inv: s = b[0] + \ldots + b[i-1] }}
while (i != n) {
    {{ s = b[0] + \ldots + b[i-1] and i != n }}
    s = s + b[i];
i = i + 1;
    {{ s = b[0] + \ldots + b[i-1] }}
}
{{ s = b[0] + \ldots + b[n-1] }}
```

- $(s = 0$ and $i = 0)$ implies $I$
- ${{I}$ and $i != n}$ S $\{I\}$?

Yes (e.g., by backward reasoning)

```java
{{ s + b[i] = b[0] + \ldots + b[i] }}
{{ s = b[0] + \ldots + b[i] }}
```
Example: sum of array

Consider the following code to compute $b[0] + \ldots + b[n-1]$:

```java
{{ b.length >= n }}

s = 0;
i = 0;
{{ Inv: s = b[0] + \ldots + b[i-1] }}
while (i != n) {
    s = s + b[i];
i = i + 1;
}
{{ s = b[0] + \ldots + b[n-1] }}
```

- $(s = 0$ and $i = 0)$ implies $I$
- $\{{ I \text{ and } i \neq n } \}$ $S$ $\{{ I \}}$
- $\{{ I \text{ and } i \geq n } \}$ implies
  $s = b[0] + \ldots + b[n-1]$?

Yes. ($I$ is the postcondition when we have $i == n$.)
Example: sum of array

Consider the following code to compute \( b[0] + \ldots + b[n-1] \):

```java
{{ b.length >= n }}
s = 0;
i = 0;
{{ Inv: s = b[0] + \ldots + b[i-1] }}
while (i != n) {
    s = s + b[i];
    i = i + 1;
}
{{ s = b[0] + \ldots + b[n-1] }}
```

- \( (s = 0 \text{ and } i = 0) \) implies I
- \( \{\{ \text{I and } i != n \} \} S \{\{ \text{I} \} \} \)
- \( \{\{ \text{I and } i == n \} \} \) implies Q

These three checks verify that the postcondition holds (i.e., the code is correct).
Termination

• Technically, this analysis does not check that the code terminates
  – it shows that the postcondition holds if the loop exits
  – but we never showed that the loop actually exits

• However, that follows from an analysis of the running time
  – e.g., if the code runs in $O(n^2)$ time, then it terminates
  – an infinite loop would be $O(\infty)$
  – any finite bound on the running time proves it terminates

• It is normal to also analyze the running time of code we write, so we get termination already from that analysis.
Example: sum of array (attempt 2)

Consider the following code to compute $b[0] + ... + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = -1;
while (i != n-1) {
    i = i + 1;
    s = s + b[i];
}
{{ s = b[0] + ... + b[n-1] }}
```
Example: sum of array (attempt 2)

Consider the following code to compute $b[0] + ... + b[n-1]$:

\[
\begin{align*}
\{\{ \text{b.length} \geq n \}\} \\
s &= 0; \\
i &= -1; \\
\{\{ \text{Inv: s = b[0] + ... + b[i]} \}\} \\
\text{while (i \neq n-1) } \{ \\
i &= i + 1; \\
s &= s + b[i]; \\
\} \\
\{\{ s = b[0] + ... + b[n-1] \}\}
\end{align*}
\]
Consider the following code to compute $b[0] + \ldots + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = -1;
{{ Inv: s = b[0] + \ldots + b[i] }}
while (i != n-1) {
    i = i + 1;
    s = s + b[i];
}
{{ s = b[0] + \ldots + b[n-1] }}
```

- $(s = 0$ and $i = -1$) implies $I$ as before
- $\{{I}$ and $i != n-1 \}}$ S $\{{I}$
  - reason backward:
    $\{{s + b[i+1] = b[0] + \ldots + b[i+1] \}}$
    $\{{s + b[i] = b[0] + \ldots + b[i] \}}$
- $(I$ and $i = n-1$) implies $Q$ as before
Example: sum of array (attempt 3)

Consider the following code to compute $b[0] + ... + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = -1;
{{ Inv: s = b[0] + ... + b[i] }}
while (i != n) {
    i = i + 1;
    s = s + b[i];
}
{{ s = b[0] + ... + b[n-1] }}
```

Suppose we use $i != n$ instead of $i != n-1$...

We can spot this bug because the postcondition no longer follows.

When $i = n$, we get:

$s = b[0] + ... + b[n]$ which is wrong
Example: sum of array (attempt 4)

Consider the following code to compute $b[0] + \ldots + b[n-1]$:

```java
{{ b.length >= n }}
s = 0;
i = -1;
{{ Inv: s = b[0] + \ldots + b[i] }}
while (i != n-1) {
    s = s + b[i];
i = i + 1;
}
{{ s = b[0] + \ldots + b[n-1] }}
```

Suppose we misorder the assignments to $i$ and $s$...

We can spot this bug because the invariant does not hold:

```java
{{ s + b[i] = b[0] + \ldots + b[i+1] }}
{{ s = b[0] + \ldots + b[i+1] }}
```

First assertion is not I.
Consider the following code to compute $\max(b[0], ..., b[n-1])$:

```java
{{ b.length >= n  and n > 0 }}
m = b[0];
i = 1;
while (i != n) {
    if (b[i] > m)
        m = b[i];
i = i + 1;
}
{{ m = max(b[0], ..., b[n-1]) }}
```
Example: max of array

Consider the following code to compute $\max(b[0], \ldots, b[n-1])$:

```java
{{ b.length >= n  and n > 0 }}
m = b[0];
i = 1;
{{ Inv: m = max(b[0], ..., b[i-1]) }}
while (i != n) {
    if (b[i] > m)
        m = b[i];
    i = i + 1;
}
{{ m = max(b[0], ..., b[n-1]) }}
```
Example: max of array

Consider the following code to compute $\max(b[0], \ldots, b[n-1])$:

```java
{{ b.length >= n  and n > 0 }}
m = b[0];
i = 1;
{{ Inv: m = max(b[0], ..., b[i-1]) }}
while (i != n) {
    if (b[i] > m)
        m = b[i];
i = i + 1;
}
{{ m = max(b[0], ..., b[n-1]) }}
```

- I holds initially: $m = \max(b[0])$
- Postcondition follows from invariant and $i = n$.
- Remains to check loop body…
Example: max of array

Consider the following code to compute \( \max(b[0], \ldots, b[n-1]) \):

\[
\begin{aligned}
&\{ \text{Inv: } m = \max(b[0], \ldots, b[i-1]) \} \\
\text{while } (i \neq n) \{ \\
&\quad \text{if } (b[i] > m) \\
&\quad \quad m = b[i]; \\
&\quad \{ \{ m = \max(b[0], \ldots, b[i]) \} \\
&\quad i = i + 1; \\
&\quad \{ \{ m = \max(b[0], \ldots, b[i-1]) \} \} \\
&\} 
\end{aligned}
\]
Example: max of array

Consider the following code to compute $\max(b[0], \ldots, b[n-1])$:

$$
\begin{align*}
\{\text{Inv: } m &= \max(b[0], \ldots, b[i-1]) \}\} \\
\text{while (i != n) } \{ \\
\quad \text{if (b[i] > m) } \\
\quad \quad m &= b[i] ; \\
\quad \text{else } \\
\quad \quad ; \\
\quad \{\text{ m = max(b[0], \ldots, b[i]) }\} \\
\qquad i &= i + 1; \\
\}
\end{align*}
$$
Example: max of array

Consider the following code to compute $\max(b[0], \ldots, b[n-1])$:

\[
\begin{align*}
\{ \text{Inv: } m &= \max(b[0], \ldots, b[i-1]) \} \\
\text{while (i != n) } &\{ \\
\text{ if (b[i] > m) } &\{ \\
\quad m &= b[i] ; \\
\quad \{ \text{ m = max(b[0], \ldots, b[i]) } \} \\
\text{ else } &\} \\
\} \\
\quad i &= i + 1 ; \\
\} 
\end{align*}
\]
Example: max of array

Consider the following code to compute $\max(b[0], \ldots, b[n-1])$:

```java
{{ Inv: m = max(b[0], ..., b[i-1]) }}
while (i != n) {
    if (b[i] > m)
        {{ b[i] = max(b[0], ..., b[i]) }}
        m = b[i];
        {{ m = max(b[0], ..., b[i]) }}
    else
        {{ m = max(b[0], ..., b[i]) }}
        i = i + 1;
}
```

CSE 331 Spring 2016
Example: max of array

Consider the following code to compute \( \max(b[0], ..., b[n-1]) \):

\[
\begin{align*}
\{ \text{Inv: } m &= \max(b[0], ..., b[i-1]) \} \\
\text{while } (i \neq n) \{ \\
& \{ (b[i] > m \text{ and } b[i] = \max(b[0], ..., b[i])) \text{ or } (b[i] \leq m \text{ and } m = \max(b[0], ..., b[i])) \} \\
& \text{if } (b[i] > m) \\
& \{ b[i] = \max(b[0], ..., b[i]) \} \\
& m = b[i] ; \\
& \text{else} \\
& \{ m = \max(b[0], ..., b[i]) \} \\
& ; \\
& i = i + 1 ; \}
\end{align*}
\]

check that I implies this… (requires some thought)
Example: max of array

Consider the following code to compute $\max(b[0], \ldots, b[n-1])$:

\[
\begin{align*}
\text{\{ Inv: m = } & \max(b[0], \ldots, b[i-1]) \}\} \\
\text{while (i != n) } & \\
\text{ \{ m = } & \max(b[0], \ldots, b[i]) \} \\
\text{if (b[i] > m) } & \\
\text{ m = b[i]; } & \\
\text{else } & \\
\text{ i = i + 1; } & \\
\end{align*}
\]

- invariant is preserved by the loop body
Example: max of array

Consider code to compute `indexOfMax(b[0], ..., b[n-1])`:

```java
{{ b.length >= n and n > 0 }}
k = 0;
m = b[k];
i = 1;
while (i != n) {
    if (b[i] > m) {
        k = i;
        m = b[k];
    }
    i = i + 1;
}
{{ m = max(b[0], ..., b[n-1]) and m = b[k] }}
```
Example: max of array

Consider code to compute \texttt{indexOfMax(b[0], \ldots, b[n-1]):}

\begin{verbatim}
{{ b.length >= n and n > 0 }}
k = 0;
m = b[k];
i = 1;
{{ Inv: m = max(b[0], \ldots, b[i-1]) and m = b[k] }}
while (i != n) {
    if (b[i] > m) {
        k = i;
        m = b[k];
    }
    i = i + 1;
}
{{ m = max(b[0], \ldots, b[n-1]) and m = b[k] }}
\end{verbatim}
Example: max of array

Consider code to compute `indexOfMax(b[0], ..., b[n-1])`:

```plaintext
{{ b.length >= n  and n > 0 }}
k = 0;
m = b[k];
i = 1;
{{ Inv: m = max(b[0], ..., b[i-1]) and m = b[k] }}
while (i != n) {
    if (b[i] > m) {
        k = i;
        m = b[k];
    }
    i = i + 1;
}
{{ m = max(b[0], ..., b[n-1]) and m = b[k] }}
```

{{ m = b[0] and k = 0 and i = 1 }}
{{ m = max(b[0], ..., b[0]) and m = b[k] }}
Example: max of array

Consider code to compute \texttt{indexOfMax}(b[0], ..., b[n-1]):

\[
\begin{aligned}
\text{\{ b.length \geq n and n > 0 \}} & \quad \text{\{ m = max(b[0], ..., b[n-1]) and m = b[k] \}} \\
\text{I holds initially}
\end{aligned}
\]

\[
\begin{aligned}
k & = 0; \\
m & = b[k]; \\
i & = 1; \\
\text{\{ Inv: m = max(b[0], ..., b[i-1]) and m = b[k] \}} \\
\text{while (i != n) } \\
\text{\{ m = max(b[0], ..., b[i-1]) and m = b[k] \}} & \quad \text{\{ m = max(b[0], ..., b[i-1]) and m = b[k] and i = n \}} \text{ implies} \\
i & = i + 1; \\
\end{aligned}
\]
Example: max of array

Consider code to compute `indexOfMax(b[0], ..., b[n-1])`:

```java
{{ b.length >= n and n > 0 }}
k = 0;
m = b[k];
i = 1;
{{ Inv: m = max(b[0], ..., b[i-1]) and m = b[k] }}
while (i != n) {
    if (b[i] > m) {
        k = i;
        m = b[k];
    }
    i = i + 1;
}
{{ m = max(b[0], ..., b[n-1]) and m = b[k] }}
```

- I holds initially
- I and i = n implies postcondition
Example: max of array

Consider code to compute \texttt{indexOfMax}(b[0], ..., b[n-1]):

\begin{verbatim}
{{ b.length >= n and n > 0 }}
k = 0;
m = b[k];
i = 1;
{{ Inv: m = max(b[0], ..., b[i-1]) and m = b[k] }}
while (i != n) {
  if (b[i] > m) {
    k = i;
m = b[k];
  }
i = i + 1;
{{ m = max(b[0], ..., b[i-1]) and m = b[k] }}
}
{{ m = max(b[0], ..., b[n-1]) and m = b[k] }}
\end{verbatim}

- I holds initially
- I and i = n implies postcondition
Example: max of array

Consider code to compute $\text{indexOfMax}(b[0], \ldots, b[n-1])$:

$\{\ \text{b.length} \geq n \ \text{and} \ n > 0 \ \}$

$k = 0$

$m = b[k]$

$i = 1$

$\{\ \text{Inv: m = max(b[0], \ldots, b[i-1]) and m = b[k]} \ \}$

while ($i != n$) {
    if ($b[i] > m$) {
        $k = i$
        $m = b[k]$
    }
    $\{\ m = \text{max(b[0], \ldots, b[i]) and m = b[k]} \ \}$
    $i = i + 1$
}

$\{\ m = \text{max(b[0], \ldots, b[n-1]) and m = b[k]} \ \}$
Example: max of array

Consider code to compute $\text{indexOfMax}(b[0], ..., b[n-1])$:

\[
\begin{align*}
\{{ \text{ b.length } \geq n \text{ and } n > 0 } \}} & \quad \text{I holds initially} \\
k &= 0; \\
m &= b[k]; \\
i &= 1; \\
\{{ \text{ Inv: } m = \text{max}(b[0], ..., b[i-1]) \text{ and } m = b[k] } \}} & \quad \text{I and } i = n \text{ implies postcondition} \\
\text{while } (i \neq n) \{ \\
\quad \text{if } (b[i] > m) \{ \\
\quad \quad k &= i; \\
\quad \quad m &= b[k]; \\
\quad \quad \{{ \text{ m = max(b[0], ..., b[i]) and m = b[k] } \}} \\
\quad \} \text{ else } \{ \\
\quad \quad \{{ \text{ m = max(b[0], ..., b[i]) and m = b[k] } \}} \\
\quad \} \\
\quad i &= i + 1; \\
\} 
\end{align*}
\]
Example: max of array

Consider code to compute $\text{indexOfMax}(b[0], \ldots, b[n-1])$:

\[
\begin{align*}
\{ & \text{ b.length }\geq n \text{ and } n > 0 \} \\
& k = 0; \\
& m = b[k]; \\
& i = 1; \\
\{ & \text{ Inv: } m = \text{max}(b[0], \ldots, b[i-1]) \text{ and } m = b[k] \} \\
\text{while } (i != n) \{ \\
& \quad \text{ if } (b[i] > m) \{ \\
& & \quad \quad k = i; \\
& & \quad \quad \{ & \text{ b[k] = max}(b[0], \ldots, b[i]) \text{ and } b[k] = b[k] \} \\
& & \quad \quad m = b[k]; \\
& & \text{ else } \{ \\
& & & \quad \{ & \text{ m = max}(b[0], \ldots, b[i]) \text{ and } m = b[k] \} \\
& & \text{ } \} \\
& \quad \} \\
& i = i + 1; \\
\} 
\]

- $I$ holds initially
- $I$ and $i = n$ implies postcondition
Example: max of array

Consider code to compute `indexOfMax(b[0], ..., b[n-1])`:

```java
{{ b.length >= n and n > 0 }}
k = 0;
m = b[k];
i = 1;
{{ Inv: m = max(b[0], ..., b[i-1]) and m = b[k] }}
while (i != n) {
    if (b[i] > m) {
        {{ b[i] = max(b[0], ..., b[i]) }}
        k = i;
        m = b[k];
    } else {
        {{ m = max(b[0], ..., b[i]) and m = b[k] }}
    }
    i = i + 1;
}
```

- I holds initially
- I and i = n implies postcondition
Consider code to compute `indexOfMax(b[0], ..., b[n-1])`:

```java
{{ b.length >= n and n > 0 }}
k = 0;
m = b[k];
i = 1;
{{ Inv: m = max(b[0], ..., b[i-1]) and m = b[k] }}
while (i != n) {
    {{ (b[i] > m) and b[i] = max(b[0], ..., b[i]) or
        (b[i] <= m) and m = max(b[0], ..., b[i]) and m = b[k] }}
    if (b[i] > m) {
        k = i;
m = b[k];
    }
i = i + 1;
}
```

• I holds initially
• I and i = n implies postcondition

Remains to show that I is stronger than this (i.e., I implies this):
• if b[i] > m = max(b[0], ..., b[i-1]),
  then b[i] = max(b[0], ..., b[i])
• if b[i] <= m = max(b[0], ..., b[i-1]),
  then m = max(b[0], ..., b[i])
Example: max of array

Consider code to compute $\text{indexOfMax}(b[0], \ldots, b[n-1])$:

$\{$
\begin{align*}
& \text{b.length} \geq n \text{ and } n > 0 \} \\
& k = 0; \\
& m = b[k]; \\
& i = 1; \\
& \{\text{Inv: m = max(b[0], \ldots, b[i-1]) and m = b[k]} \} \\
& \text{while } (i \neq n) \{ \\
& \quad \text{if } (b[i] > m) \{ \\
& \quad \quad k = i; \\
& \quad \quad m = b[k]; \\
& \quad \} \\
& \quad i = i + 1; \\
& \} \\
& \{\text{m = max(b[0], \ldots, b[n-1]) and m = b[k]} \}
\end{align*}$

- I holds initially
- I and $i = n$ implies postcondition
- I holds after loop body
Example: partition array

Consider the following code to put the negative values at the beginning of array $b$:

```cpp
{{ }}
i = k = 0;
while (i != n) {
    if (b[i] < 0) {
        swap b[i], b[k];
        k = k + 1;
    }
    i = i + 1;
}

{{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[n-1] }}
```

(Also: $b$ contains the same numbers since we use swaps.)
Example: partition array

Consider the following code to put the negative values at the beginning of array b:

```c
{{ }}
i = k = 0;
{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
    if (b[i] < 0) {
        swap b[i], b[k];
        k = k + 1;
    }
    i = i + 1;
}
{{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[n-1] }}
```
Example: partition array

Consider the following code to put the negative values at the beginning of array $b$:

```
{{
  i = k = 0;
}

{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
  if (b[i] < 0) {
    swap b[i], b[k];
    k = k + 1;
  }
  i = i + 1;
}

{{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[n-1] }}
```

- $I$ holds initially:
  - $b[0], ..., b[-1]$ is empty
- $I$ and $i = n$ implies postcondition
Example: partition array

Consider the following code to put the negative values at the beginning of array b:

```c
{{ }}
i = k = 0;
{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
    if (b[i] < 0) {
        swap b[i], b[k];
        k = k + 1;
    }
    i = i + 1;
}
{{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[n-1] }}
```

- I holds initially
- I and i = n implies postcondition
Example: partition array

Consider the following code to put the negative values at the beginning of array \( b \):

\[
\begin{align*}
&\{\} \\
i &= k = 0; \\
&\{\text{Inv: } b[0], \ldots, b[k-1] < 0 \leq b[k], \ldots, b[i-1]\} \\
\text{while (}i \neq n\text{) } \{ \\
&\quad \text{if (}b[i] < 0\text{) } \{ \\
&\qquad \text{swap } b[i], b[k]; \\
&\qquad k = k + 1; \\
&\quad \} \\
&\quad i = i + 1; \\
&\} \\
&\{\text{b[0], ..., b[k-1] < 0 \leq b[k], ..., b[i-1]}\} \\
\end{align*}
\]

- \( I \) holds initially
- \( I \) and \( i = n \) implies postcondition
Example: partition array

Consider the following code to put the negative values at the beginning of array $b$:

```c
{{ }}
i = k = 0;
{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
    if (b[i] < 0) {
        swap b[i], b[k];
        k = k + 1;
    }
    {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i] }}
    i = i + 1;
}
{{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[n-1] }}
```

- $I$ holds initially
- $I$ and $i = n$ implies postcondition
Example: partition array

Consider the following code to put the negative values at the beginning of array \( b \):

\[
\begin{align*}
\{ & \} \\
i = k = 0; \\
\{ \text{ Inv: } b[0], \ldots, b[k-1] < 0 \iff b[k], \ldots, b[i-1] \} \\
\text{while } (i \neq n) \{ \\
\hspace{1em} \{ b[0], \ldots, b[k-1] < 0 \iff b[k], \ldots, b[i-1] \} \\
\hspace{1em} \text{if } (b[i] < 0) \{ \\
\hspace{2em} \text{swap } b[i], b[k]; \\
\hspace{2em} k = k + 1; \\
\} \\
\hspace{1em} \{ b[0], \ldots, b[k-1] < 0 \iff b[k], \ldots, b[i] \} \\
\hspace{1em} i = i + 1; \\
\} \\
\{ \text{ b[0], \ldots, b[k-1] < 0 \iff b[k], \ldots, b[n-1] } \}
\end{align*}
\]

- \( I \) holds initially
- \( I \) and \( i = n \) implies postcondition
Example: partition array

Consider the following code to put the negative values at the beginning of array $b$:

```plaintext
{{ }}
i = k = 0;
{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
    if (b[i] < 0) {
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] and b[i] < 0 }}
        swap b[i], b[k];
        k = k + 1;
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i] }}
    } else {
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] and b[i] >= 0 }}
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i] }}
    }
    i = i + 1;
}
```

- $I$ holds initially
- $I$ and $i = n$ implies postcondition
Example: partition array

Consider the following code to put the negative values at the beginning of array b:

```c
{{ }}
i = k = 0;
{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
    if (b[i] < 0) {
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] and b[i] < 0 }}
        swap b[i], b[k];
        k = k + 1;
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i] }}
    } else {
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] and b[i] >= 0 }}
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i] }}
    }
    i = i + 1;
}
• I holds initially
• I and i = n implies postcondition
```
Example: partition array

Consider the following code to put the negative values at the beginning of array \( b \):

\[
\begin{align*}
\{\} & \quad \text{I holds initially} \\
i &= k = 0;
\{\text{Inv: } b[0], ..., b[k-1] \leq b[k], ..., b[i-1]\} & \quad \text{I and } i = n \text{ implies postcondition} \\
\text{while } (i \neq n) \{ & \\
\quad \text{if } (b[i] < 0) \{ & \\
\quad \quad \{\begin{align*}
\{& b[0], ..., b[k-1] \leq b[k], ..., b[i-1] \text{ and } b[i] < 0 \} \\
\text{swap } b[i], b[k]; & \\
& k = k + 1; \\
\{& b[0], ..., b[k-1] \leq b[k], ..., b[i]\}
\end{align*}\} & \quad \text{Remain to check this…}
\}
\quad i = i + 1;
\}\}
\{b[0], ..., b[k-1] < 0 \leq b[k], ..., b[n-1]\}
\end{align*}
\]
Example: partition array

Consider the following code to put the negative values at the beginning of array $b$:

```
{{ }}
i = k = 0;
{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
    if (b[i] < 0) {
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] and b[i] < 0 }}
        swap b[i], b[k];
        k = k + 1;
        {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i] }}
    }
    i = i + 1;
}
{{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[n-1] }}
```

- $I$ holds initially
- $I$ and $i = n$ implies postcondition
Example: partition array

Consider the following code to put the negative values at the beginning of array b:

```plaintext
{{ }}
i = k = 0;
{{ Inv: b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] }}
while (i != n) {
  if (b[i] < 0) {
    {{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[i-1] and b[i] < 0 }}
    swap b[i], b[k];
    {{ b[0], ..., b[k] < 0 <= b[k+1], ..., b[i] }}
    k = k + 1;
  }
  i = i + 1;
}
{{ b[0], ..., b[k-1] < 0 <= b[k], ..., b[n-1] }}
```

- I holds initially
- I and i = n implies postcondition

This is a valid triple.
(Takes some thought.)
Example: partition array

Consider the following code to put the negative values at the beginning of array \( b \):

\[
\begin{align*}
&\{\text{ }\} \\
i &= k = 0; \\
&\{\text{ Inv: } b[0], \ldots, b[k-1] < 0 \leq b[k], \ldots, b[i-1] \} \\
&\text{while} \ (i \neq n) \ {\} \\
&\quad \{ \text{ if } (b[i] < 0) \ {\} \\
&\quad \quad \{ \text{ swap } b[i], b[k]; \}
&\quad \quad \{ k = k + 1; \}
&\quad \} \\
&\quad \{ i = i + 1; \}
&\} \\
&\{ b[0], \ldots, b[k-1] < 0 \leq b[k], \ldots, b[n-1] \}
\end{align*}
\]

- I holds initially
- I and \( i = n \) implies postcondition
- I holds after loop body