Arrays
Assignment of Values of an Array


Assignment may work as before: \[
\langle P[A[x]/v] \rangle \\
v = A[x] \\
\langle P \rangle \quad \text{assignment}
\]

But a complication can occur: \[
\langle A[y] = 0 \rangle \\
A[x] = 1; \\
\langle A[y] = 0 \rangle \quad ???
\]

The conclusion is not valid if \( x = y \).

A correct rule must account for possible changes to A[y], A[z+3], etc., when A[x] changes.
Our solution: Treat an assignment to an array value

\[ A[e_1] = e_2 \]

as an assignment of the whole array:

\[ A = A\{e_1 \leftarrow e_2\} ; \]

where the term “\( A\{e_1 \leftarrow e_2\} \)” denotes an array identical to \( A \) except the \( e_1^{th} \) element is changed to have the value \( e_2 \).
Array Assignment: Definition and Examples

**Definition:** $A\{i \leftarrow e\}$ denotes the array with entries given by

$$A\{i \leftarrow e\}[j] = \begin{cases} 
  e, & \text{if } j = i \\
  A[j], & \text{if } j \neq i 
\end{cases}.$$ 

**Examples:**

$$A\{1 \leftarrow 7\}{\{2 \leftarrow 14\}}[2] = ??$$

$$A\{1 \leftarrow 7\}{\{2 \leftarrow 14\}{\{3 \leftarrow 21\}}}[2] = ??$$

$$A\{1 \leftarrow 7\}{\{2 \leftarrow 14\}{\{3 \leftarrow 21\}}}[i] = ??$$
The Array-Assignment Rule

Array assignment:

\[
\left( \frac{Q[A\{e_1 \leftarrow e_2\}/A]}{A[e_1] = e_2} \right) (Array \ assignment)
\]

where

\[
A\{i \leftarrow e\}[j] = \begin{cases} 
  e, & \text{if } j = i \\
  A[j], & \text{if } j \neq i
\end{cases}
\]
Example

Prove the following is satisfied under partial correctness.

$$\langle A[x] = x_0 \land A[y] = y_0 \rangle$$

$t = A[x]$ ;


$A[y] = t$ ;

$$\langle A[x] = y_0 \land A[y] = x_0 \rangle$$

We do assignments bottom-up, as always....
Example: push up assertions for assignments

\[
( A[x] = x_0 \land A[y] = y_0 )
\]

\[
t = A[x] ;
\]

\[
\]

\[
( A\{y \leftarrow t\}[x] = y_0 \land A\{y \leftarrow t\}[y] = x_0 )
\]

\[
A[y] = t ;
\]

\[
( A[x] = y_0 \land A[y] = x_0 ) \quad \text{array assignment}
\]
Example: push up assertions for assignments

\[(A[x] = x_0 \land A[y] = y_0)\]

t = A[x] ;
\[(A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0
\land A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0)\]

\[(A\{y \leftarrow t\}[x] = y_0 \land A\{y \leftarrow t\}[y] = x_0)\] array assignment

A[y] = t ;
\[(A[x] = y_0 \land A[y] = x_0)\] array assignment
Example: push up assertions for assignments

\( \langle A[x] = x_0 \land A[y] = y_0 \rangle \)
\( \langle A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}\}[x] = y_0 \)
\( \land A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}\}[y] = x_0 \rangle \)

\( t = A[x] ; \)
\( \langle A\{x \leftarrow A[y]\}\{y \leftarrow t\}\}[x] = y_0 \)
\( \land A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0 \rangle \)

\( A[x] = A[y] ; \)

\( \langle A\{y \leftarrow t\}[x] = y_0 \land A\{y \leftarrow t\}[y] = x_0 \rangle \)

\( A[y] = t ; \)

\( \langle A[x] = y_0 \land A[y] = x_0 \rangle \)
Example: push up assertions for assignments

\[
\begin{align*}
& (A[x] = x_0 \land A[y] = y_0) \\
& \quad \land A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[x] = y_0 \\
& \quad \land A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[y] = x_0
\end{align*}
\]

implied (a)

t = A[x] ;

\[
\begin{align*}
& (A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0 \\
& \quad \land A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0
\end{align*}
\]

assignment


\[
\begin{align*}
& (A\{y \leftarrow t\}[x] = y_0 \land A\{y \leftarrow t\}[y] = x_0)
\end{align*}
\]

array assignment

A[y] = t ;

\[
\begin{align*}
& (A[x] = y_0 \land A[y] = x_0)
\end{align*}
\]

array assignment
Example: Proof of implied

As “implied (a)”, we need to prove the following.

**Lemma:**

\[
A\{ x \leftarrow A[y]\}\{ y \leftarrow A[x]\}\}[x] = A[y]
\]

and

\[
A\{ x \leftarrow A[y]\}\{ y \leftarrow A[x]\}\}[y] = A[x].
\]

**Proof.**

In the second equation, the index element is the assigned element.

For the first equation, we consider two cases.

- If \( y \neq x \), the “\( \{ y \leftarrow \ldots \} \)” is irrelevant, and the claim holds.
- If \( y = x \), the result on the left is \( A[x] \), which is also \( A[y] \).
Example: Alternative proof

For an alternative proof, use the definition of \( M \{ i \leftarrow e \}[j] \), with \( A \{ x \leftarrow A[y] \} \) as \( M \), \( i = y \) and \( e = A[x] \):

\[
A \{ x \leftarrow A[y] \} \{ y \leftarrow A[x] \}[j] = \begin{cases} 
A[x], & \text{if } y = j \\
A \{ x \leftarrow A[y] \}[j], & \text{if } y \neq j 
\end{cases}
\]

At index \( j = y \), this is just \( A[x] \), as required.

In the case \( j = x \), we get the required value \( A[y] \). (Why?)

And, finally, if \( j \neq x \) and \( j \neq y \), then

\[
\]

as we should have required.
Example: reversing an array

**Example:** Given an array $R$ with $n$ elements, reverse the elements.

**Algorithm:** exchange $R[j]$ with $R[n + 1 - j]$, for each $1 \leq j \leq \lfloor n/2 \rfloor$.

A possible program is

```c
j = 1;
while ( 2*j <= n ) {
    t = R[j] ;
    R[j] = R[n+1-j] ;
    R[n+1-j] = t ;
    j = j + 1 ;
}
```

Needed: a postcondition, and a loop invariant.
Reversal code: conditions and an invariant

Precondition: \( (\forall x \left( (1 \leq x \leq n) \rightarrow (R[x] = r_x) \right) ) \).

Postcondition: \( (\forall x \left( (1 \leq x \leq n) \rightarrow (R[x] = r_{n+1-x}) \right) ) \).

Invariant? When exchanging at position \( j \)?

- If \( x < j \) or \( x > n + 1 - j \), then \( R[x] \) and \( R[n + 1 - x] \) have already been exchanged.
- If \( j \leq x \leq n + 1 - j \), then no exchange has happened yet.

Thus let \( \text{Inv}'(j) \) be the formula

\[
\left( \forall x \left( (1 \leq x < j) \rightarrow (R[x] = r_{n+1-x} \land R[n + 1 - x] = r_x) \right) \land (j \leq x \leq n/2) \rightarrow (R[x] = r_x \land R[n + 1 - x] = r_{n+1-x}) \right) .
\]

and \( \text{Inv}(j) = \text{Inv}'(j) \land (1 \leq j \leq n) \).
Reversal: Annotations around the loop

The annotations surrounding the while-loop:

\[
\left\{ \left( n \geq 0 \right) \wedge \left( \forall x \left( (1 \leq x \leq n) \rightarrow (R[x] = r_x) \right) \right) \right\} \quad \left( \text{Implied (a)} \right)
\]

\[
\left\{ \text{Inv}(1) \right\} \quad \left( \text{Assignment} \right)
\]

\[
\text{j = 1 ;}
\]

\[
\left\{ \text{Inv}(j) \right\} \quad \left( \text{Partial-While} \right)
\]

while ( 2*j <= n ) {

\[
\left\{ \left( \text{Inv}(j) \wedge (2j \leq n) \right) \right\} \quad \left( \text{(TBA)} \right)
\]

\[
\vdots
\]

\[
\left\{ \text{Inv}(j) \right\} \quad \left( \text{Partial-While} \right)
\]

\[
\left\{ \left( \text{Inv}(j) \wedge (2j > n) \right) \right\} \quad \left( \text{Implied (b)} \right)
\]

\[
\left\{ \left( \forall x \left( (1 \leq x \leq n) \rightarrow (R[x] = r_{n+1-x}) \right) \right) \right\}
\]
Reversal code: annotations inside the loop

We must now handle the code inside the loop.

\[
\begin{align*}
& \text{(Inv}(j) \land 2j \leq n) \} \\
& \text{Inv}(j + 1)[R'/R], \text{ where } R' \text{ is} \\
& \quad \{ R[j \leftarrow R[n + 1 - j]\} \{ (n + 1 - j) \leftarrow R[j]\} \} \\
& t = R[j]; \ R[j] = R[n+1-j]; \ R[n+1-j] = t; \\
& \text{Lemma} \\
& j = j + 1; \\
& \text{assignment}
\end{align*}
\]
Proof of Implied Condition (c)

Recall $\text{Inv}'(j)$:

$$
\left( \forall x \left( \left( (1 \leq x < j) \rightarrow (R[x] = r_{n+1-x} \land R[n + 1 - x] = r_x) \right) \right.
\left. \land \left( (j \leq x \leq n/2) \rightarrow (R[x] = r_x \land R[n + 1 - x] = r_{n+1-x}) \right) \right) \right) .
$$

We need this to imply $\text{Inv}'(j + 1)[R' / R]$, which is

$$
\left( \forall x \left( \left( (1 \leq x < j + 1) \rightarrow (R'[x] = r_{n+1-x} \land R'[n + 1 - x] = r_x) \right) \right.
\left. \land \left( (j + 1 \leq x \leq n/2) \rightarrow (R'[x] = r_x \land R'[n + 1 - x] = r_{n+1-x}) \right) \right) \right) ,
$$

which by the construction of $R'$ is equivalent to

$$
\left( \forall x \left( \left( (1 \leq x < j) \rightarrow (R[x] = r_{n+1-x} \land R[n + 1 - x] = r_x) \right) \right.
\left. \land \left( R'[j] = r_{n+1-j} \right) \land \left( R'[n + 1 - j] = r_j \right) \right.
\left. \land \left( (j + 1 \leq x \leq n/2) \rightarrow (R[x] = r_x \land R[n + 1 - x] = r_{n+1-x}) \right) \right) \right) .
$$
Example: Binary Search
Binary Search

Binary search is a very common technique, to find whether a given item exists in a sorted array.

Although the algorithm is simple in principle, it is easy to get the details wrong. Hence verification is in order.

Inputs: Array $A$ indexed from 1 to $n$; integer $x$.

Precondition: $A$ is sorted: $\forall i \forall j \left( (1 \leq i < j \leq n) \rightarrow (A[i] \leq A[j]) \right)$.

Output values: boolean $\text{found}$; integer $m$.

Postcondition: Either $\text{found}$ is true and $A[m] = x$, or $\text{found}$ is false and $x$ does not occur at any location of $A$.

(Also, $A$ and $x$ are unchanged; We simply won’t write to either.)
Code: The outer loop

\( \forall i \forall j \left((1 \leq i < j \leq n) \rightarrow (A[i] \leq A[j])\right) \)

\[ I \]
\[
\begin{align*}
\text{while} \ (\ l \leq u \ \text{and} \ \neg \text{found} \ ) \ {\{ & \\
\text{\hspace{1em}} \ (I \wedge (l \leq u \wedge \neg \text{found})) & \text{partial-while} \\
\text{\hspace{1em}} \ m = (l+u) \text{ div } 2 & \\
\text{\hspace{1em}} \ (J) & \\
\text{\hspace{1em}} \text{if} \ (A[m] = x) \ {\{ & \\
\text{\hspace{2em}} \text{...Body omitted...} & \\
\text{\hspace{1em}} \}} & \\
\text{\hspace{1em}} } & \text{if-then-else} \\
\text{\hspace{1em}} \}} & \\
\end{align*}
\]
\[
(I \wedge \neg(l \leq u \wedge \neg \text{found})) \}
\[
((\text{found} \wedge A[m] = x) \lor (\neg \text{found} \wedge \forall k \neg (A[k] = x))) \}
\]

Program Verification Example: Binary Search
Code: the if-statement

\[ J \]

if ( A[m] = x ) {

\[ J \land (A[m] = x) \] if-then-else
found = true;

\[ I \]

} else if ( A[m] < x ) {

\[ J \land \neg(A[m] = x) \land (A[m] < x) \] if-then-else
l = m+1;

\[ I \]

} else {

\[ J \land \neg(A[m] = x) \land \neg(A[m] < x) \] if-then-else
u = m - 1;

\[ I \]
}

\[ I \] if-then-else
Invariant for Binary Search

In the loop, there are two cases:

- We have found the target, at position \( m \).
- We have not yet found the target; if it is present, it must lie between \( A[\ell] \) and \( A[u] \) (inclusive).

Expressed as a formula:

\[
(found \land A[m] = x) \lor (\neg found \land \forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right))
\]

It turns out that the above is more specific than necessary. As the actual invariant, we shall use the formula

\[
I = (found \rightarrow A[m] = x) \land (\forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right))
\]

(Exercise: As you go through the proof, check what would happen if we used the first formula instead.)
Annotations for while

\[ \forall i \forall j \left( 1 \leq i < j \leq n \rightarrow (A[i] \leq A[j]) \right) \]

\[ l = 1; \quad u = n; \quad \text{found} = \text{false}; \]

\[ \left( \text{found} \rightarrow A[m] = x \right) \land \left( \forall i \left( (A[i] = x) \rightarrow (l \leq i \leq u) \right) \right) \]

while ( \( l \leq u \) \&\& \! \text{found} ) {

\[ \left( \text{found} \rightarrow A[m] = x \right) \land \left( \forall i \left( (A[i] = x) \rightarrow (l \leq i \leq u) \right) \right) \]

\[ \land (l \leq u) \land \neg\text{found} \] while

\[ \left( \forall i \left( (A[i] = x) \rightarrow (l \leq i \leq u) \right) \right) \land \neg\text{found} \land (l \leq \lfloor (l + u)/2 \rfloor \leq u) \]

implied

\[ m = (l+u) \text{ div } 2; \]

\[ \left( \forall i \left( (A[i] = x) \rightarrow (l \leq i \leq u) \right) \right) \land \neg\text{found} \land (l \leq m \leq u) \]

assignment

The last condition is the formula “\( J \)” : the precondition for the
if-statement.
First Branch of the if-Statement

\[
\{ \forall i \left( (A[i] = x) \rightarrow (\ell \leq i \leq u) \right) \land \neg \text{found} \land (\ell \leq m \leq u) \} \\
\text{if ( A[m] = x )} \{ \\
\{ \forall i \left( (A[i] = x) \rightarrow (\ell \leq i \leq u) \right) \land \neg \text{found} \land (\ell \leq m \leq u) \land (A[m] = x) \} \\
\quad \text{if-then-else} \\
\{ (\text{true} \rightarrow A[m] = x) \land \left( \forall i \left( (A[i] = x) \rightarrow (\ell \leq i \leq u) \right) \right) \} \\
\quad \text{implied} \\
\text{found = true; } \\
\{ (\text{found} \rightarrow A[m] = x) \land \left( \forall i \left( (A[i] = x) \rightarrow (\ell \leq i \leq u) \right) \right) \} \\
\quad \text{assignment} \\
\}
\]

The implication is trivial.
Second Branch of the if-Statement

\[ \{( \forall i ((A[i] = x) \rightarrow (\ell \leq i \leq u)) \land \neg\text{found} \land (\ell \leq m \leq u) \land \neg(A[m] = x) \} \]

if (A[m] < x) {
\[ \{( \forall i ((A[i] = x) \rightarrow (\ell \leq i \leq u)) \land \neg\text{found} \land (\ell \leq m \leq u) \land \neg(A[m] = x) \land (A[m] < x) \} \]
\[ \{( \text{found} \rightarrow A[m] = x) \land (\forall i ((A[i] = x) \rightarrow (m + 1 \leq i \leq u)) \} \}

1 = m + 1;
\[ \{( \text{found} \rightarrow A[m] = x) \land (\forall i ((A[i] = x) \rightarrow (\ell \leq i \leq u)) \} \]

To justify the implication, show that \( A[j] < x \) whenever \( \ell \leq j \leq m \).

This follows from the condition that \( A \) is sorted, together with \( A[m] < x \).
An Extended Example: Sorting
Postcondition for Sorting

Suppose the code $C_{\text{sort}}$ is intended to sort $n$ elements of array $A$.

Give pre- and postconditions for $C_{\text{sort}}$, using a predicate $\text{sorted}(A, n)$ which is true iff $A[1] \leq A[2] \leq ... \leq A[n]$.

**First Attempt**

\[ (n \geq 1) \]

\[ C_{\text{sort}} \]

\[ (\text{sorted}(A, n)) \]
Postcondition for Sorting

Suppose the code $C_{\text{sort}}$ is intended to sort $n$ elements of array $A$.

Give pre- and postconditions for $C_{\text{sort}}$, using a predicate $\text{sorted}(A, n)$ which is true iff $A[1] \leq A[2] \leq \ldots \leq A[n]$.

**First Attempt**

\[
\begin{align*}
\{ & n \geq 1 \} & \{ & n \geq 1 \} \\
C_{\text{sort}} & \quad \text{for } i = 1 \text{ to } n \{ \\
& \quad A[i] = 0 ; \\
& \quad \} \\
\{ & \text{sorted}(A, n) \} & \{ & \text{sorted}(A, n) \} 
\end{align*}
\]
Let $\text{permutation}(A, A', n)$ mean that array $A[1], A[2], \ldots, A[n]$ is a permutation of array $A'[1], A'[2], \ldots, A'[n]$.

($A'$ will be a logical variable, not a program variable.)

**Second Attempt**

\[
\{ n \geq 1 \land A = A' \}
\]

$C_{\text{sort}}$

\[
\{ \text{sorted}(A, n) \land \\
\text{permutation}(A, A', n) \}
\]
Postcondition for Sorting, II

Let \(\text{permutation}(A, A', n)\) mean that array \(A[1], A[2], \ldots, A[n]\) is a permutation of array \(A'[1], A'[2], \ldots, A'[n]\).

\(A'\) will be a logical variable, not a program variable.

**Second Attempt**

\[
\{ n \geq 1 \land A = A' \} \\
\{ n \geq 1 \land A = A' \}
\]

\(C_{\text{sort}}\)

\[
\text{some algorithm on } A ; \\
n = 1 ;
\]

\[
\{ \text{sorted}(A, n) \land \text{permutation}(A, A', n) \} \\
\{ \text{sorted}(A, n) \land \text{permutation}(A, A', n) \}
\]
Final Attempt (Correct)

\[
\begin{align*}
\langle n \geq 1 \land n = n_0 \land A = A' \rangle \\
C_{\text{sort}} \\
\langle \text{sorted}(A, n_0) \land \text{permutation}(A, A', n_0) \rangle
\end{align*}
\]
Algorithms for Sorting

We shall briefly describe two algorithms for sorting.

- Insertion Sort
- Quicksort

Each has an “inner loop” which we will then consider.
Overview of Insertion Sort


Plan: insert each element, in turn, into the array of previously sorted elements.

Algorithm:

At the start, $A[1]$ is sorted (as an array of length 1).  
For each $k$ from 2 to $n$ 

Assume the array is sorted up to position $k-1$  

Compare it with $A[k-1], A[k-2], \text{etc.}$, until its proper place is reached.
Insertion Sort: Inserting one element

Possible code for the insertion loop:

```c
i = k;
while (i > 1) {
    if (A[i] < A[i-1]) {
        t = A[i];
        A[i] = A[i-1];
        A[i-1] = t;
    }
    i = i - 1;
}
```

For correctness of this code, see the current assignment.
Overview of Quicksort

Quicksort is an ingenious algorithm, with many variations. Sometimes it works very well, sometimes not so well. We shall ignore most of those issues, however, and just look at a central step of the algorithm.

Idea:

Select one element of the array, called the *pivot*. (Which one? A complicated issue. YMMV.)

Separate the array into two parts: those less than or equal to the pivot, and those greater than the pivot.

Recursively sort each of the two parts.

Here, we shall focus on the middle step: “partition” the array according to the chosen pivot.
Partitioning an Array

Given: Array $X$ of length $n$, and a pivot $p$.

Goal: Put the “small” elements (those less than or equal to $p$) to the left part of the array, and the “large” elements (those greater than $p$) to the right.

Plan: Scan the array. Upon finding a large element appearing before a small element, exchange the two.

Requisite: Do all exchanges in a single scan. (Linear time!)
Partition: The Algorithm

Idea: keep the array elements in three sections of the array.

- Those known to be small (less than or equal to the pivot).
- Those known to be large (larger than the pivot).
- Unknown elements (not yet examined).

We mark the separations with pointers (indices) $a$ and $b$, as shown.

\[
\begin{array}{c|c|c}
\text{small} & \text{large} & \text{unknown} \\
\hline
a & b & \end{array}
\]

One step of the algorithm:

- If $X[b]$ is small, swap it with $X[a]$ and increment $a$.
- Increment $b$. 
\[ \{ \ n \geq 1 \ \} \]
a = 1;
while ( a < n && X[a] \leq p ) {
    a = a + 1;
}
b = a + 1;
while ( b <= n ) {
    if ( X[b] \leq p ) {
        // Swap if needed
        t = X[b]; X[b] = X[a]; X[a] = t;
        a = a + 1;
    }
    b = b + 1;
}
\[ \{ \ \exists z \left( 1 \leq z \leq n + 1 \right) \land (X[1..z] \leq p) \land (X[z..n] > p) \} \ \]
Annotation: First loop

Desired postcondition for the first loop:

\[
\| (X[1..a] \leq p) \land ((a \geq n) \lor (X[a] > p)) \| .
\]

Annotation for the while, and pushing up, yields

\[
\| (X[1..1] \leq p) \| \\
a = 1 ; \\
\| (X[1..a] \leq p) \| \\
while ( a < n && X[a] \leq p ) 
\begin{cases}
\| (X[1..a] \leq p) \land ((a < n) \land (X[a] \leq p)) \| \\
\| (X[1..a + 1] \leq p) \| \\
a = a + 1 ; \\
\| (X[1..a] \leq p) \| 
\end{cases}
\]
The Second while-Loop

For the second while-loop, a good candidate for the invariant is

\[(X[1..a] \leq p) \land (X[a..b] > p)\]  .

Let’s see if this works....

Colour key:

- **Greenish**: lower part of the array
- **Blueish**: upper part of the array
- **Either, reddened**: result of a substitution
- **Reddish**: condition from guards
Inside the while-Loop

“And” the loop guard to the invariant at the start of the loop. Then pushing up through the assignment and if yields

\[
\{ \ (X[1..a] \leq p) \land (X[a..b] > p) \land (b \leq n) \ \} \\
\text{if} \ (X[b] \leq p) \ \{ \\
\{ \ (X[1..a] \leq p) \land (X[a..b] > p) \land (b \leq n) \land X[b] \leq p \ \} \\
\quad \text{if-then} \\
\} \\
\{ \ (X[1..a] \leq p) \land (X[a..b + 1] > p) \ \} \\
\quad \text{if-then + implied} \\
b = b + 1 \\
\{ \ (X[1..a] \leq p) \land (X[a..b] > p) \ \} \\
\quad \text{assignment}
Inside the if-Statement

Push up for the assignments inside the if:

\[
\begin{align*}
&\text{if-then} & (X[1..a) \leq p) \land (X[a..b) > p) \land (b \leq n) \land X[b] \leq p \\
&\text{implied} & ((X[1..a) \leq p)) \land (X[a] > p) \land (X[a + 1..b) > p) \land X[b] \leq p \\
&t = X[b] ; X[b] = X[a] ; X[a] = t ; & ((X[1..a) \leq p)) \land (X[a] \leq p) \land (X[a + 1..b) > p) \land X[b] > p \\
&\text{swap} & (X[1..a + 1) \leq p) \land (X[a + 1..b + 1) > p) \\
&\text{implied} & (X[1..a + 1) \leq p) \land (X[a + 1..b + 1) > p) \\
&a = a + 1 ; & a + 1 = a + 1 \\
&\text{assignment} & (X[1..a) \leq p) \land (X[a..b + 1) > p) \\
\end{align*}
\]

(The extra “implied” just makes the “swap” clearer.)
Putting It All Together

The annotation thus far works fine. But there is a “glitch”....

Between the loops, we have

\[
\begin{align*}
&\quad (X[1..a] \leq p) \land ((a \geq n) \lor (X[a] > p)) \quad \text{partial-while} \\
&\quad (X[1..a] \leq p) \land (X[a..a + 1] > p) \quad \text{implied} \\
&\quad b = a + 1 \\
&\quad (X[1..a] \leq p) \land (X[a..b] > p) \quad \text{assignment}
\end{align*}
\]

But the “implied” fails in the case that the first loop ended with \( a = n \) — we can’t deduce \( X[a] > p \).

Solution: either

- add an extra test to the code, or

- add \( 1 \leq a \leq n \) to the first invariant, and modify the second to \( (X[1..a] \leq p) \land ((a = n) \lor (X[a..b] > p)) \).
Second while-Loop: Full annotation

\[
\begin{align*}
& (1 \leq a \leq n) \land (X[1..a] \leq p) \land ((a \geq n) \lor (X[a] > p)) \downarrow \text{partial-while} \\
& (X[1..a] \leq p) \land ((a = n) \lor (X[a..a + 1] > p)) \downarrow \text{implied} \\
& b = a + 1 ; \\
& (X[1..a] \leq p) \land ((a = n) \lor (X[a..b] > p)) \downarrow \text{assignment} \\
\text{while ( b \leq n ) } \{ \\
& (X[1..a] \leq p) \land ((a = n) \lor (X[a..b] > p)) \land (b \leq n) \downarrow \text{partial-while} \\
& (X[1..a] \leq p) \land (X[a..b] > p) \land (b \leq n) \downarrow \text{implied} \\
\text{if (X[b] \leq p ) } \{ \\
& \} \\
& (X[1..a] \leq p) \land (X[a..b] > p) \downarrow \text{if-then} \\
& (X[1..a] \leq p) \land ((a = n) \lor (X[a..b] > p)) \downarrow \text{implied} \\
\} \\
& (X[1..a] \leq p) \land ((a = n) \lor (X[a..b] > p)) \land (b > n) \downarrow \text{partial-while} \\
& \exists z ((1 \leq z \leq n + 1) \land (X[1..z] \leq p) \land (X[z..n] > p)) \downarrow \text{implied} \\
\end{align*}
\]

“Implied” proofs left to you.