Arrays

Assignment of Values of an Array

Let `A` be an array of `n` integers: `A[1]`, `A[2]`, ..., `A[n]`.

Assignment may work as before:

$$\{\!|\, P[A[x]/v] \,|\!\}$$
`v = A[x] ;`
$$\{\!|\, P \,|\!\}$$
assignment

But a complication can occur:

$$\{\!|\, A[y] = 0 \,|\!\}$$
`A[x] = 1;`
$$\{\!|\, A[y] = 0 \,|\!\}$$???

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.

Assignment to a Whole Array

Our solution: Treat an assignment to an array value

$$\texttt{A[}\mathsf{e}_1\texttt{]} \texttt{ = } \mathsf{e}_2$$

as an assignment of the whole array:

$$\texttt{A = A}\{\mathsf{e}_1 \leftarrow \mathsf{e}_2\} \texttt{ ;}$$

where the term "$\texttt{A}\{\mathsf{e}_1 \leftarrow \mathsf{e}_2\}$" 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:

$$\frac{}{\{\!|\, Q[A\{e_1 \leftarrow e_2\}/A] \,|\!\}\ \texttt{A[e}_1\texttt{]} \texttt{ = e}_2\ \{\!|\, Q \,|\!\}}\text{(Array assignment)}$$

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.

$$\{\!|\, A[x] = x_0 \wedge A[y] = y_0 \,|\!\}$$

```
t = A[x] ;
```

```
A[x] = A[y] ;
```

```
A[y] = t ;
```

$$\{\!|\, A[x] = y_0 \wedge A[y] = x_0 \,|\!\}$$

We do assignments bottom-up, as always....

Example: push up assertions for assignments

$$\{\!|\ A[x] = x_0 \wedge A[y] = y_0\ |\!\}$$

```
t = A[x] ;
```

```
A[x] = A[y] ;
```

$$\{\!|\ A\{y \leftarrow t\}[x] = y_0 \wedge A\{y \leftarrow t\}[y] = x_0\ |\!\}$$

```
A[y] = t ;
```

$$\{\!|\ A[x] = y_0 \wedge A[y] = x_0\ |\!\}$$

array assignment

Example: push up assertions for assignments

$$(\!| A[x] = x_0 \wedge A[y] = y_0 |\!)$$

```
t = A[x] ;
```

$$(\!| A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0 \wedge A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0 |\!)$$

```
A[x] = A[y] ;
```

$$(\!| A\{y \leftarrow t\}[x] = y_0 \wedge A\{y \leftarrow t\}[y] = x_0 |\!)$$ array assignment

```
A[y] = t ;
```

$$(\!| A[x] = y_0 \wedge A[y] = x_0 |\!)$$ array assignment

Example: push up assertions for assignments

$$(\!| A[x] = x_0 \wedge A[y] = y_0 |\!)$$

$$(\!| A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[x] = y_0 \wedge A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[y] = x_0 |\!)$$

```
t = A[x] ;
```

$$(\!| A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0 \wedge A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0 |\!)$$ assignment

```
A[x] = A[y] ;
```

$$(\!| A\{y \leftarrow t\}[x] = y_0 \wedge A\{y \leftarrow t\}[y] = x_0 |\!)$$ array assignment

```
A[y] = t ;
```

$$(\!| A[x] = y_0 \wedge A[y] = x_0 |\!)$$ array assignment

Example: push up assertions for assignments

$(\!| A[x] = x_0 \wedge A[y] = y_0 |\!)$

$(\!| A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[x] = y_0$
$\quad \wedge A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[y] = x_0 |\!)$ — implied (a)

```
t = A[x] ;
```

$(\!| A\{x \leftarrow A[y]\}\{y \leftarrow t\}[x] = y_0$
$\quad \wedge A\{x \leftarrow A[y]\}\{y \leftarrow t\}[y] = x_0 |\!)$ — assignment

```
A[x] = A[y] ;
```

$(\!| A\{y \leftarrow t\}[x] = y_0 \wedge A\{y \leftarrow t\}[y] = x_0 |\!)$ — array assignment

```
A[y] = t ;
```

$(\!| A[x] = y_0 \wedge A[y] = x_0 |\!)$ — 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

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

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

$$A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[j] = A[j] \ ,$$

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

```
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: $\Big(\forall x \Big((1 \leq x \leq n) \to (R[x] = r_x)\Big)\Big)$.

Postcondition: $\Big(\forall x \Big((1 \leq x \leq n) \to (R[x] = r_{n+1-x})\Big)\Big)$.

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 *Inv'*(j) be the formula

$$\Big(\forall x \Big(\Big((1 \leq x < j) \to (R[x] = r_{n+1-x} \wedge R[n + 1 - x] = r_x)\Big) \\ \wedge \Big((j \leq x \leq n/2) \to (R[x] = r_x \wedge R[n + 1 - x] = r_{n+1-x})\Big)\Big)\Big) \ .$$

and *Inv*$(j) =$ *Inv'*$(j) \wedge (1 \leq j \leq n)$.

Reversal: Annotations around the loop

The annotations surrounding the `while`-loop:

$\{\!|\ \big((n \geq 0) \wedge \big(\forall x \big((1 \leq x \leq n) \rightarrow (R[x] = r_x)\big)\big)\big)\ |\!\}$

$\{\!|\ \mathit{Inv}(1)\ |\!\}$ — implied (a)

```
j = 1 ;
```

$\{\!|\ \mathit{Inv}(j)\ |\!\}$ — assignment

```
while ( 2*j <= n ) {
```

$\{\!|\ (\mathit{Inv}(j) \wedge (2j \leq n))\ |\!\}$ — partial-`while`

$\vdots$

$\{\!|\ \mathit{Inv}(j)\ |\!\}$ — *(TBA)*

```
}
```

$\{\!|\ (\mathit{Inv}(j) \wedge (2j > n))\ |\!\}$ — partial-`while`

$\{\!|\ \big(\forall x \big((1 \leq x \leq n) \rightarrow (R[x] = r_{n+1-x})\big)\big)\ |\!\}$ — implied (b)

Reversal code: annotations inside the loop

We must now handle the code inside the loop.

$\{\!|\ (\mathit{Inv}(j) \wedge 2j \leq n)\ |\!\}$ partial-`while`

$\{\!|\ \mathit{Inv}(j+1)[R'/R]$, *where* R' *is*

$R\{j \leftarrow R[n+1-j]\}\{(n+1-j) \leftarrow R[j]\}\ |\!\}$ implied (c)

```
t = R[j]; R[j] = R[n+1-j]; R[n+1-j] = t;
```

$\{\!|\ \mathit{Inv}(j+1)\ |\!\}$ Lemma

```
j = j + 1 ;
```

$\{\!|\ \mathit{Inv}(j)\ |\!\}$ assignment

Proof of Implied Condition (c)

Recall *Inv'*(j):

$$\Big(\forall x \Big(\big((1 \le x < j) \to (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)\big) \\ \wedge \big((j \le x \le n/2) \to (R[x] = r_x \wedge R[n+1-x] = r_{n+1-x})\big)\Big)\Big) \ .$$

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

$$\Big(\forall x \Big(\big((1 \le x < j+1) \to (R'[x] = r_{n+1-x} \wedge R'[n+1-x] = r_x)\big) \\ \wedge \big((j+1 \le x \le n/2) \to (R'[x] = r_x \wedge R'[n+1-x] = r_{n+1-x})\big)\Big)\Big) \ ,$$

which by the construction of R' is equivalent to

$$\Big(\forall x \Big(\big((1 \le x < j) \to (R[x] = r_{n+1-x} \wedge R[n+1-x] = r_x)\big) \\ \wedge (R'[j] = r_{n+1-j}) \wedge (R'[n+1-j] = r_j) \\ \wedge \big((j+1 \le x \le n/2) \to (R[x] = r_x \wedge R[n+1-x] = r_{n+1-x})\big)\Big)\Big) \ .$$

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 `found`; integer m.

Postcondition: Either `found` is true and $A[m] = x$, or `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

$\{\!|\ \forall i\ \forall j \left((1 \le i < j \le n) \to (A[i] \le A[j])\right)\ |\!\}$

```
l = 1;   u = n;   found = false;
```

$\{\!|\ I\ |\!\}$

```
while ( l <= u and !found ) {
```

$\{\!|\ I \wedge (l \le u \wedge \neg found)\ |\!\}$ partial-while

```
  m = (l+u) div 2;
```

$\{\!|\ J\ |\!\}$

```
  if (A[m] = x) {
```

...Body omitted...

```
  }
```

$\{\!|\ I\ |\!\}$ if-then-else

```
}
```

$\{\!|\ I \wedge \neg(l \le u \wedge \neg found)\ |\!\}$ partial-while

$\{\!|\ (found \wedge A[m] = x) \vee (\neg found \wedge \forall k\ \neg(A[k] = x))\ |\!\}$

Code: the `if`-statement

$\{\!|\ J\ |\!\}$

```
if ( A[m] = x ) {
```

  $\{\!|\ J \wedge (A[m] = x)\ |\!\}$ if-then-else

```
  found = true;
```

  $\{\!|\ I\ |\!\}$

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

  $\{\!|\ J \wedge \neg(A[m] = x) \wedge (A[m] < x)\ |\!\}$ if-then-else

```
  l = m+1;
```

  $\{\!|\ I\ |\!\}$

```
} else {
```

  $\{\!|\ J \wedge \neg(A[m] = x) \wedge \neg(A[m] < x)\ |\!\}$ 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 beween $A[\ell]$ and $A[u]$ (inclusive).

Expressed as a formula:

$$(\mathit{found} \wedge A[m] = x) \vee \Big(\neg \mathit{found} \wedge \forall i \Big((A[i] = x) \rightarrow (\ell \leq i \leq u)\Big)\Big) \ .$$

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

$$I = (\mathit{found} \rightarrow A[m] = x) \wedge \Big(\forall i \Big((A[i] = x) \rightarrow (\ell \leq i \leq u)\Big)\Big) \ .$$

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

Annotations for `while`

$\{\!|\ \forall i\ \forall j \left((1 \le i < j \le n) \to (A[i] \le A[j])\right)\ |\!\}$

```
l = 1;    u = n;    found = false;
```

$\{\!|\ (\mathit{found} \to A[m] = x) \wedge \left(\forall i \left((A[i] = x) \to (\ell \le i \le u)\right)\right)\ |\!\}$

```
while ( l <= u && !found ) {
```

$\{\!|\ (\mathit{found} \to A[m] = x) \wedge \left(\forall i \left((A[i] = x) \to (\ell \le i \le u)\right)\right) \wedge (l \le u) \wedge \neg \mathit{found}\ |\!\}$ partial-while

$\{\!|\ \forall i \left((A[i] = x) \to (\ell \le i \le u)\right) \wedge \neg \mathit{found} \wedge (\ell \le \lfloor (\ell + u)/2 \rfloor \le u)\ |\!\}$ implied

```
  m = (l+u) div 2 ;
```

$\{\!|\ \forall i \left((A[i] = x) \to (\ell \le i \le u)\right) \wedge \neg \mathit{found} \wedge (\ell \le m \le u)\ |\!\}$ assignment

The last condition is the formula "J": the precondition for the `if`-statement.

First Branch of the `if`-Statement

$\{\!|\ \forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right) \wedge \neg \mathit{found} \wedge (\ell \leq m \leq u)\ |\!\}$

```
if ( A[m] = x ) {
```

$\quad \{\!|\ \forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right) \wedge \neg \mathit{found} \wedge (\ell \leq m \leq u) \wedge (A[m] = x)\ |\!\}$ if-then-else

$\quad \{\!|\ (\mathit{true} \rightarrow A[m] = x) \wedge \left(\forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right)\right)\ |\!\}$ implied

```
  found = true;
```

$\quad \{\!|\ (\mathit{found} \rightarrow A[m] = x) \wedge \left(\forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right)\right)\ |\!\}$ assignment

```
}
```

The implication is trivial.

Second Branch of the `if`-Statement

$$\{\!|\ \forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right) \wedge \neg \mathit{found} \wedge (\ell \leq m \leq u) \wedge \neg(A[m] = x)\ |\!\}$$

```
if ( A[m] < x ) {
```

$$\{\!|\ \forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right) \wedge \neg \mathit{found} \wedge (\ell \leq m \leq u) \wedge \neg(A[m] = x) \wedge (A[m] < x)\ |\!\}$$ if-then-else

$$\{\!|\ (\mathit{found} \rightarrow A[m] = x) \wedge \left(\forall i \left((A[i] = x) \rightarrow (m + 1 \leq i \leq u)\right)\right)\ |\!\}$$ implied

```
  l = m + 1;
```

$$\{\!|\ (\mathit{found} \rightarrow A[m] = x) \wedge \left(\forall i \left((A[i] = x) \rightarrow (\ell \leq i \leq u)\right)\right)\ |\!\}$$ assignment

```
}
```

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 $C_{\texttt{sort}}$ is intended to sort n elements of array A.

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

First Attempt

$$\{\!|\ n \geq 1\ |\!\}$$

$$C_{\texttt{sort}}$$

$$\{\!|\ \textit{sorted}(A, n)\ |\!\}$$

Postcondition for Sorting

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

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

First Attempt

$(\!| \, n \geq 1 \, |\!)$

$C_{\texttt{sort}}$

$(\!| \, \textit{sorted}(A, n) \, |\!)$

$(\!| \, n \geq 1 \, |\!)$

```
for i = 1 to n {
  A[i] = 0 ;
}
```

$(\!| \, \textit{sorted}(A, n) \, |\!)$

Postcondition for Sorting, II

Let *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

$(\!| \, n \geq 1 \wedge A = A' \, |\!)$

$C_{\texttt{sort}}$

$(\!| \, \mathit{sorted}(A, n) \wedge$
$\mathit{permutation}(A, A', n) \, |\!)$

Postcondition for Sorting, II

Let *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

$(\!| \, n \geq 1 \wedge A = A' \, |\!)$

$C_{\texttt{sort}}$

$(\!| \, \mathit{sorted}(A, n) \wedge \mathit{permutation}(A, A', n) \, |\!)$

$(\!| \, n \geq 1 \wedge A = A' \, |\!)$

some algorithm on A ;
`n = 1 ;`

$(\!| \, \mathit{sorted}(A, n) \wedge \mathit{permutation}(A, A', n) \, |\!)$

Postcondition for Sorting, III

Final Attempt (Correct)

$$\{\!|\, n \geq 1 \wedge n = n_0 \wedge A = A' \,|\!\}$$

$$C_{\texttt{sort}}$$

$$\{\!|\, \mathit{sorted}(A, n_0) \wedge \mathit{permutation}(A, A', n_0) \,|\!\}$$

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

Input: Array A, with indices $A[1] \ldots A[n]$.

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 $k-1$
> > Insert $A[k]$ into its correct place among $A[1] \ldots A[k-1]$:
> >
> > > Compare it with $A[k-1]$, $A[k-2]$, etc., until its proper place is reached.

Insertion Sort: Inserting one element

Possible code for the insertion loop:

```
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.

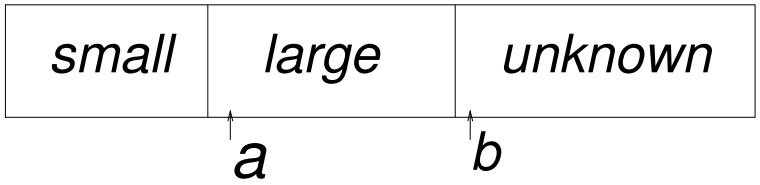
Requisite: Do all exchanges in a single scan. (Linear time!)

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.



One step of the algorithm:

– If $X[b]$ is small, swap it with $X[a]$ and increment a.
– Increment b.

Code and Pre- and Postconditions

$\{\!|\ n \geq 1\ |\!\}$

```
a = 1 ;
while ( a < n && X[a] <= p ) {          // Initialize
  a = a + 1 ;
}
b = a + 1 ;
while ( b <= n ) {
  if ( X[b] <= p ) {                    // Swap if needed
    t = X[b] ; X[b] = X[a] ; X[a] = t ;
    a = a + 1 ;
  }
  b = b + 1 ;
}
```

$\{\!|\ \exists z \Big((1 \leq z \leq n+1) \wedge (X[1..z) \leq p) \wedge (X[z..n] > p) \Big)\ |\!\}$

Notation: "$X[j..k)\,\ldots$" means "$X[i]\,\ldots$, for each $j \leq i < k$".
"$X[j..k]\,\ldots$" means "$X[i]\,\ldots$, for each $j \leq i \leq k$".

Annotation: First loop

Desired postcondition for the first loop:

$$\{\!|\ (X[1..a) \leq p) \wedge \Big((a \geq n) \vee (X[a] > p)\Big)\ |\!\} \ .$$

Annotation for the `while`, and pushing up, yields

$\{\!|\ (X[1..1) \leq p)\ |\!\}$ — implied

```
a = 1 ;
```

$\{\!|\ (X[1..a) \leq p)\ |\!\}$ — assignment

```
while ( a < n && X[a] <= p ) {
```

$\{\!|\ (X[1..a) \leq p) \wedge \Big((a < n) \wedge (X[a] \leq p)\Big)\ |\!\}$ — partial-`while`

$\{\!|\ (X[1..a+1) \leq p)\ |\!\}$ — implied

```
  a = a + 1 ;
```

$\{\!|\ (X[1..a) \leq p)\ |\!\}$ — assignment

```
}
```

$\{\!|\ (X[1..a) \leq p) \wedge \Big((a \geq n) \vee (X[a] > p)\Big)\ |\!\}$ — partial-`while`

The Second `while`-Loop

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

$$(X[1..a) \leq p) \wedge (X[a..b) > p) \quad .$$

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

$(\!|\ (X[1..a) \leq p) \wedge (X[a..b) > p) \wedge (b \leq n)\ |\!)$

```
if (X[b] <= p ) {
```

$(\!|\ (X[1..a) \leq p) \wedge (X[a..b) > p) \wedge (b \leq n) \wedge X[b] \leq p\ |\!)$

if-then

```
}
```

$(\!|\ (X[1..a) \leq p) \wedge (X[a..b+1) > p)\ |\!)$

if-then + implied

```
b = b + 1
```

$(\!|\ (X[1..a) \leq p) \wedge (X[a..b) > p)\ |\!)$ assignment

Inside the `if`-Statement

Push up for the assignments inside the `if`:

$\{\!|\ (X[1..a) \leq p) \wedge (X[a..b) > p) \wedge (b \leq n) \wedge X[b] \leq p\ |\!\}$

if-then

$\{\!|\ ((X[1..a) \leq p)) \wedge (X[a] > p) \wedge (X[a+1..b) > p) \wedge X[b] \leq p\ |\!\}$

implied

```
t = X[b] ; X[b] = X[a] ; X[a] = t ;
```

$\{\!|\ ((X[1..a) \leq p)) \wedge (X[a] \leq p) \wedge (X[a+1..b) > p) \wedge X[b] > p\ |\!\}$

swap

$\{\!|\ (X[1..a+1) \leq p) \wedge (X[a+1..b+1) > p)\ |\!\}$

implied

```
a = a + 1 ;
```

$\{\!|\ (X[1..a) \leq p) \wedge (X[a..b+1) > p)\ |\!\}$ assignment

(*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

$$\begin{array}{ll} \{\!| \ (X[1..a) \leq p) \wedge \Big((a \geq n) \vee (X[a] > p)\Big) \ |\!\} & \text{partial-}\texttt{while} \\ \{\!| \ (X[1..a) \leq p) \wedge (X[a..a+1) > p) \ |\!\} & \text{implied} \end{array}$$

```
b = a + 1 ;
```

$$\{\!| \ (X[1..a) \leq p) \wedge (X[a..b) > p) \ |\!\} \qquad \text{assignment}$$

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 $(X[1..a) \leq p) \wedge \Big((a = n) \vee (X[a..b) > p)\Big)$.

Second `while`-Loop: Full annotation

$\{\!|\ (1 \le a \le n) \wedge (X[1..a) \le p) \wedge \Big((a \ge n) \vee (X[a] > p)\Big)\ |\!\}$ partial-`while`

$\{\!|\ (X[1..a) \le p) \wedge \Big((a = n) \vee (X[a..a+1) > p)\Big)\ |\!\}$ implied

```
b = a + 1 ;
```

$\{\!|\ (X[1..a) \le p) \wedge \Big((a = n) \vee (X[a..b) > p)\Big)\ |\!\}$ assignment

```
while ( b <= n ) {
```

$\{\!|\ (X[1..a) \le p) \wedge \Big((a = n) \vee (X[a..b) > p)\Big) \wedge (b \le n)\ |\!\}$ partial-`while`

$\{\!|\ (X[1..a) \le p) \wedge (X[a..b) > p) \wedge (b \le n)\ |\!\}$ implied

```
  if (X[b] <= p ) {

    }
```

$\{\!|\ (X[1..a) \le p) \wedge (X[a..b) > p)\ |\!\}$ `if-then`

$\{\!|\ (X[1..a) \le p) \wedge \Big((a = n) \vee (X[a..b) > p)\Big)\ |\!\}$ implied

```
}
```

$\{\!|\ (X[1..a) \le p) \wedge \Big((a = n) \vee (X[a..b) > p)\Big) \wedge (b > n)\ |\!\}$ partial-`while`

$\{\!|\ \exists z \Big((1 \le z \le n+1) \wedge (X[1..z) \le p) \wedge (X[z..n] > p)\Big)\ |\!\}$ implied

"Implied" proofs left to you.