Array Element Assignments

CS 536: Science of Programming, Fall 2019

11/10

A. Why?

- Array assignments aren't like assignments to plain variables because the actual item to change can't be determined until runtime. We can handle this by extending our notion of assignment and/or substitution.

B. Outcomes

After this lecture, you should

- Know how to perform textual substitution to replace an array element.
- Know how to calculate the *wp* of an array element assignment.

C. Array Element Assignments

- An array assignment $\texttt{b}[e_0] := e_1$ (where e_0 and e_1 are expressions) is different from a plain variable assignment because the exact element being changed may not be known at program annotation time. E.g., compare these two triples:
  - **Valid**: $\{\mathbf{T}\}\ \texttt{x := y; y := y+1}\ \{\texttt{x} < \texttt{y}\}$
  - **Invalid**: $\{\mathbf{T}\}\ \texttt{b[i] := b[j]; b[j] := b[j]+1}\ \{\texttt{b[i]} < \texttt{b[j]}\}$
- The problem is what happens if $\texttt{i} = \texttt{j}$ at runtime: What is

$$wp(\texttt{b[j] := b[j]+1}, \texttt{b[i]} < \texttt{b[j]})\quad ?$$

- The answer should be something like

"If $\texttt{i} \neq \texttt{j}$ then $\texttt{b[i]} < \texttt{b[j]+1}$, else $\texttt{b[j]+1} < \texttt{b[j]+1}$ (which is false)"

- There are two alternatives for handling array assignments
- The alternative we'll use involves defining the *wp* of an array assignment using an extended notion of textual substitution:

$$wp(\texttt{b}[e_0] := e_1, p) \equiv p[e_1/\texttt{b}[e_0]] \text{ and } \{\, p[e_1/\texttt{b}[e_0]\,]\,\}\ \texttt{b}[e_0] := e_1\ \{\, p \,\}$$

- Of course, we need to figure out what syntactic substitution for an array indexing expression means: $(predicate)[expression/\texttt{b}[e_0]]$
- Side note: The other way to handle array assignments, the Dijkstra / Gries technique, is to introduce a new kind of expression and view the array assignment $\texttt{b}[e_0] := e_1$ as short for $\texttt{b} :=$ this new kind of expression.
- If we have time later (we probably won't), we'll study this.

D. Substitution for Array Elements

- We'll need to substitute into expressions and predicates. We'll tackle expressions first; the new cases are
  - $(\texttt{b}_2[e_2])[e/\texttt{b}_1[e_1]]$ where $\texttt{b}_1$ and $\texttt{b}_2$ are different arrays
  - $(\texttt{b}_1[e_2])[e/\texttt{b}_1[e_1]]$

- If $\mathtt{b}_1$ and $\mathtt{b}_2$ are different array names then substituting into $\mathtt{b}_2[e_2]$ will only require us to look at substituting into e_2:

$$(\mathtt{b}_2[e_2])[e/\mathtt{b}_1[e_1]] \equiv \mathtt{b}_2[e_2{}'] \text{ where } e_2{}' \equiv (e_2)[e/\mathtt{b}_1[e_1]]$$

- When the the array names match, as in $(\mathtt{b[i]})[e'/\mathtt{b}[e]]$, we have to check the indexes `i` and e for equality at runtime; to do that, we can use a conditional expression.
- **Definition**: $(\mathtt{b[i]})[e_1/\mathtt{b}[e]] \equiv (\mathbf{if}\ \mathtt{i} = e\ \mathbf{then}\ e_1\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi})$.
  - If $\mathtt{i} = e$ at runtime, then $(\mathtt{b[i]})[e_1/\mathtt{b}[e]] = e_1$.
  - If $\mathtt{i} \neq e$ at runtime, then $(\mathtt{b[i]})[e_1/\mathtt{b}[e]] = \mathtt{b[i]}$.
- **Example 1**: $(\mathtt{b[i]})[5/\mathtt{b[0]}] \equiv (\mathbf{if}\ \mathtt{i} = 0\ \mathbf{then}\ 5\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi})$.
- **Example 2**: $(\mathtt{b[i]})[e_1/\mathtt{b[j]}] \equiv (\mathbf{if}\ \mathtt{i} = \mathtt{j}\ \mathbf{then}\ e_1\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi})$.
- **Example 3**: $(\mathtt{b[i]})[\mathtt{b[j]+1}/\mathtt{b[j]}] \equiv (\mathbf{if}\ \mathtt{i} = \mathtt{j}\ \mathbf{then}\ \mathtt{b[j]+1}\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi})$.
  - Note: In $(\mathtt{b[i]})[e_1/\mathtt{b}[e_0]]$, we don't substitute into e_1, even if it involves `b`.
- **Example 4**: $(\mathtt{b[i]})[\mathtt{b[k]}/\mathtt{b[j]}] \equiv (\mathbf{if}\ \mathtt{i} = \mathtt{j}\ \mathbf{then}\ \mathtt{b[k]}\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi})$.

The general case for array element substitution

- More generally, for $(\mathtt{b}[e_2])[e/\mathtt{b}[e_2]]$, where e_2 is not just a simple variable or constant, we have to check e_2 for substitutions.
- **Definition**: $(\mathtt{b}[e_2])[e_0/\mathtt{b}[e_1]] \equiv (\mathbf{if}\ e_2{}' = e_1\ \mathbf{then}\ e_0\ \mathbf{else}\ \mathtt{b}[e_2{}']\ \mathbf{fi})$ where $e_2{}' \equiv (e_2)[e_0/\mathtt{b}[e_1]]$.
- This subsumes the earlier case, since if $e_2 \equiv \mathtt{i}$ then $e_2{}' \equiv \mathtt{i}[e_0/\mathtt{b}[e_1]] \equiv \mathtt{i}$. We get

$$(\mathtt{b[i]})[e/\mathtt{b}[e_1]] \equiv (\mathbf{if}\ \mathtt{i} = e_1\ \mathbf{then}\ e\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi})$$

- **Example 5**: Consider $(\mathtt{b[b[0]]})[5/\mathtt{b[0]}]$ — how should it behave? The nested `b[i]` should behave like `5` if `i = 0`, otherwise it's behaves like `b[i]` as usual. The outer `b[…]` should behave like `5` if its inner index behaves like `0`, otherwise it should behave as `b`[its inner index].
- Following the definition above, we get

  $(\mathtt{b[b[i]]})[5/\mathtt{b[0]}] \equiv (\mathbf{if}\ e_2{}' = 0\ \mathbf{then}\ 5\ \mathbf{else}\ \mathtt{b}[e_2{}']\ \mathbf{fi})$
  where $e_2{}' \equiv (\mathtt{b[i]})[5/\mathtt{b[0]}] \equiv (\mathbf{if}\ \mathtt{i} = 0\ \mathbf{then}\ 5\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi})$
- Substituting the (textual) value of $e_2{}'$ gives us

$$\begin{aligned}&(\mathtt{b[b[i]]})[5/\mathtt{b[0]}]\\ \equiv\ &\mathbf{if}\ (\mathbf{if}\ \mathtt{i} = 0\ \mathbf{then}\ 5\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi}) = 0\\ &\quad \mathbf{then}\ 5\\ &\quad \mathbf{else}\ \mathtt{b}[\mathbf{if}\ \mathtt{i} = 0\ \mathbf{then}\ 5\ \mathbf{else}\ \mathtt{b[i]}\ \mathbf{fi}]\ \mathbf{fi}\end{aligned}$$

- In the next section, we'll see how to simplify expressions so that we like this to get

$$\mathbf{if}\ \mathtt{i} = 0\ \mathbf{then}\ \mathtt{b[5]}\ \mathbf{else}\ \mathbf{if}\ \mathtt{b[i]} = 0\ \mathbf{then}\ 5\ \mathbf{else}\ \mathtt{b[b[i]]}\ \mathbf{fi}\ \mathbf{fi}$$

E. Optimization of Static Cases

- Because $e[e_1/b[e_0]]$ can result in a complicated piece of text, it can be useful to shorten it using various optimizations, similarly to how compilers can optimize code. All the optimizations below are intended to be done "statically" (at compile time) — we inspect the text of an expression before the code ever runs.

- For the easiest examples, if we know whether $\mathtt{i} = e_0$ or not, then we can optimize **if** $\mathtt{i} = e_0$ **then** … **else** **fi** to just the true branch or the false branch.
- **Example 6**: We can optimize $(\mathtt{b[0]})[e_1/\mathtt{b[2]}] \equiv$ **if** $0 = 2$ **then** e_1 **else** b[0] **fi** to just b[0].
- **Example 7**: Similarly, we can optimize $(\mathtt{b[2]})[e_1/\mathtt{b[2]}] \equiv$ **if** $2 = 2$ **then** e_1 **else** b[2] **fi** to just e_1.
- So let's use the following optimizations:
  - If we know statically that $\mathtt{i} = e$, then let $(\mathtt{b[i]})[e'/\mathtt{b}[e]] \equiv e'$.
  - If we know statically that $\mathtt{i} \neq e$, then let $(\mathtt{b[i]})[e'/\mathtt{b}[e]] \equiv \mathtt{b[i]}$.
- **Example 8**:
  - For $(\mathtt{b[0]})[e'/\mathtt{b[1]}]$, use b[0], not (**if** $0 = 1$ **then** e' **else** b[0] **fi**).
  - For $(\mathtt{b[1]})[e'/\mathtt{b[1]}]$, use e', not (**if** $1 = 1$ **then** e' **else** b[0] **fi**).
  - For $(\mathtt{b[1]})[3/\mathtt{b[2]}]$, use b[1], not (**if** $1 = 2$ **then** 3 **else** b[1] **fi**).
  - For $(\mathtt{b[x]})[e'/\mathtt{b[x]}]$, use e', not (**if** $\mathtt{x} = \mathtt{x}$ **then** e' **else** b[x] **fi**).

F. Rules for Simplifying Conditional Expressions

- Let’s identify some general rules for simplifying conditional expressions and predicates involving them. This will let us simplify calculation of *wp* for array assignments.
  - (**if** T **then** e_1 **else** e_2 **fi**) $= e_1$ and (**if** F **then** e_1 **else** e_2 **fi**) $= e_2$
  - (**if** B **then** e **else** e **fi**) $= e$
  - If $(B \rightarrow e_1 = e_2)$, then (**if** B **then** e_1 **else** e_2 **fi**) $= e_2$
  - If $(\neg B \rightarrow e_1 = e_2)$, then (**if** B **then** e_1 **else** e_2 **fi**) $= e_1$
  - Let $\ominus$ be a unary operator or relation and $\oplus$ be a binary operation or relation
    - $\ominus$(**if** B **then** e_1 **else** e_2 **fi**) = (**if** B **then** $\ominus\, e_1$ **else** $\ominus\, e_2$ **fi**)
    - (**if** B **then** e_1 **else** e_2 **fi**) $\oplus\, e_3$ = (**if** B **then** $e_1 \oplus e_3$ **else** $e_2 \oplus e_3$ **fi**)
  - b[**if** B **then** e_1 **else** e_2 **fi**] = **if** B **then** b[e_1] **else** b[e_2] **fi**
  - For any function f(…), f(**if** B **then** e_1 **else** e_2 **fi**) = **if** B **then** f(e_1) **else** f(e_2) **fi**
- If B, B_1, and B_2 are boolean expressions, then
  - (**if** B **then** B_1 **else** F **fi**) $\Leftrightarrow (B \wedge B_1)$
  - (**if** B **then** F **else** B_2 **fi**) $\Leftrightarrow (\neg B \wedge B_2)$
  - (**if** B **then** B_1 **else** T **fi**) $\Leftrightarrow (B \rightarrow B_1) \Leftrightarrow (\neg B \vee B_1) \Leftrightarrow (B \wedge B_1)$
  - (**if** B **then** T **else** B_2 **fi**) $\Leftrightarrow (\neg B \rightarrow B_2) \Leftrightarrow (B \vee B_2)$
  - (**if** B **then** B_1 **else** B_2 **fi**) $\Leftrightarrow ((B \rightarrow B_1) \wedge (\neg B \rightarrow B_2)) \Leftrightarrow ((B \wedge B_1) \vee (\neg B \wedge B_2))$.
- **Example 9**:

  wp(b[j] **:=** b[j]+1, b[i] < b[j])

  $\equiv$ (b[i] < b[j])[b[j]+1/b[j]]

  $\equiv$ (b[i]) [b[j]+1/b[j]] < (b[j]) [b[j]+1/b[j]]

  $\equiv$ **if** i = j **then** b[j]+1 **else** b[i] **fi** < b[j]+1

  $\Leftrightarrow$ **if** i = j **then** b[j]+1 < b[j]+1 **else** b[i] < b[j]+1 **fi**

$\Leftrightarrow$ **if** i = j **then** F **else** b[i] < b[j]+1 **fi**

$\Leftrightarrow$ i $\neq$ j $\wedge$ b[i] < b[j]+1

So we know {i $\neq$ j $\wedge$ b[i] < b[j]+1} b[j] **:=** b[j]+1 {b[i] < b[j]}

G. Swapping Array Elements

- To illustrate the use of array references, let's look at the problem of swapping array elements.
- To swap simple variables x and y using a temporary variable u, we can use logical variables c and d and prove

  {x = c $\wedge$ y = d} u **:=** x**;** x **:=** y**;** y **:=** u {x = d $\wedge$ y = c}

- We can prove this program correct by expanding to a full proof outline; here we're using *wp*.

  {x = c $\wedge$ y = d}
  {y = d $\wedge$ x = c} u **:=** x;
  {y = d $\wedge$ u = c} x **:=** y;
  {x = d $\wedge$ u = c} y **:=** u
  {x = d $\wedge$ y = c}

- **Example 10**: For swapping b[m] and b[n], we want to prove

  {b[m] = c $\wedge$ b[n] = d} u **:=** b[m]**;** b[m] **:=** b[n]**;** b[n] **:=** u {b[m] = d $\wedge$ b[n] = c}

  As with simple variables, we can prove this holds by using *wp* to expand to the full proof outline

  {p} {q_3} u **:=** b[m]**;** {q_2} b[m] **:=** b[n]**;** {q_1} b[n] **:=** u {q}

  where

  - $p \equiv$ b[m] = c $\wedge$ b[n] = d
  - $q \equiv$ b[m] = d $\wedge$ b[n] = c
  - $q_1 \equiv wp$(b[n] **:=** u, q) $\equiv q$ [u/b[n]],
  - $q_2 \equiv wp$(b[m] **:=** b[n], q_1) $\equiv q_1$ [b[n]/b[m]]
  - $q_3 \equiv wp$(u **:=** b[m], q_2) $\equiv q_2$ [b[m]/u]
  - and (we hope), $p \rightarrow q_3$.

  We'll do this in steps.

  $q_1 \equiv q$ [u/b[n]]
  $\equiv$ (b[m] = d $\wedge$ b[n] = c) [u/b[n]]
  $\equiv$ (b[m] = d)[u/b[n]] $\wedge$ (b[n] = c)[u/b[n]]
  $\equiv$ (b[m])[u/b[n]] = d $\wedge$ (b[n])[u/b[n]] = c
  $\equiv$ (**if** m = n **then** u **else** b[m] **fi**) = d $\wedge$ u = c   // Stop here for a purely syntactic result

  $q_2 \equiv q_1$ [b[n]/b[m]]
  $\equiv$ ((**if** m = n **then** u **else** b[m] **fi**) = d $\wedge$ u = c) [b[n]/b[m]]
  $\equiv$ (**if** m = n **then** u **else** (b[m]) [b[n]/b[m]] **fi**) = d $\wedge$ u = c
  $\equiv$ (**if** m = n **then** u **else** b[n] **fi**) = d $\wedge$ u = c

$q_3 \equiv q_2[\mathtt{b[m]/u}]$

$\equiv ((\mathbf{if}\ \mathtt{m = n}\ \mathbf{then}\ \mathtt{u}\ \mathbf{else}\ \mathtt{b[n]}\ \mathbf{fi}) = \mathtt{d} \wedge \mathtt{u = c})[\mathtt{b[m]/u}]$

$\equiv (\mathbf{if}\ \mathtt{m = n}\ \mathbf{then}\ \mathtt{b[m]}\ \mathbf{else}\ \mathtt{b[n]}\ \mathbf{fi}) = \mathtt{d} \wedge \mathtt{b[m] = c})$

// Continuing with logical manipulation

$\Leftrightarrow (\mathbf{if}\ \mathtt{m = n}\ \mathbf{then}\ \mathtt{b[n]}\ \mathbf{else}\ \mathtt{b[n]}\ \mathbf{fi}) = \mathtt{d} \wedge \mathtt{b[m] = c})$ // if $\mathtt{m = n}$ then $\mathtt{b[m] = b[n]}$

$\Leftrightarrow \mathtt{b[n] = d} \wedge \mathtt{b[m] = c}$

Since $p \equiv \mathtt{b[m] = c} \wedge \mathtt{b[n] = d}$, we get $p \rightarrow q_3$. (End of **Example 10**)

Array Element Assignments

CS 536: Science of Programming

A. Why?

- Array assignments aren't like assignments to plain variables because the actual item to change can't be determined until runtime. We can handle this by extending our notion of assignment and/or substitution.

B. Outcomes

At the end of this activity you should:

- Be able to perform textual substitution to replace an array element.
- Be able to calculate the *wp* of an array element assignment.

C. Questions

For each of the questions below, calculate the given weakest precondition. Then try logically simplifying it to something easier to read.

1. Calculate *wp*(`b[0] := 9`, $x > b[i]$).
2. Calculate *wp*(`b[i] := b[j]`, $b[k] = 0$).
3. Calculate *wp*(`b[i] := b[j]`, $b[j] = z$).
4. Calculate *wp*(`b[i] := 1`, $b[i] = b[j]$).
5. Calculate *wp*(`b[i] := x; b[j] := y`, $b[i] \neq b[j]$).
6. Calculate *wp*(`b[i] := x`, $b[b[i]] \neq b[i]$).
7. Is the triple $\{i < b[i] < b[j]\}$ `b[b[i]] := b[j]` $\{b[i] \neq b[j])\}$ valid?
8. Define a predicate function swapped(b_1, b_2, i, j) that yields true iff b_1 and b_2 are equal except their values at i and j are swapped.
9. Give a recursive definition for a predicate function permuted(b_1, b_2) that yields true iff b_1 and b_2 have exactly the same multi-set of values, in possibly different orders. E.g., permuted([1, 3, 1, 2], [1, 2, 1, 3]) = T but permuted([1, 3, 1, 2], [1, 3, 3, 2]) = F. One way to approach the problem is that permuted(b_1, b_2) is true if $b_1 = b_2$ or if you can perform a sequence of swaps on (say) b_1 to get b_2.

Solution to Activity 20 (Array Element Assignments)

1. $wp(\texttt{b[0]} := 9,\ \texttt{x} > \texttt{b[i]}) \equiv (\texttt{x} > \texttt{b[i]})[9/\texttt{b[0]}]$

   $\equiv \texttt{x} > \textbf{if } \texttt{i} = 0 \textbf{ then } 9 \textbf{ else } \texttt{b[i]} \textbf{ fi}$

   $\Leftrightarrow \textbf{if } \texttt{i} = 0 \textbf{ then } \texttt{x} > 9 \textbf{ else } \texttt{x} > \texttt{b[i]} \textbf{ fi}$

   $\Leftrightarrow (\texttt{i} = 0 \wedge \texttt{x} > 9) \vee (\texttt{i} \neq 0 \wedge \texttt{x} > \texttt{b[i]})$

2. $wp(\texttt{b[i]} := \texttt{b[j]},\ \texttt{b[k]} = 0) \equiv (\texttt{b[k]} = 0)[\texttt{b[j]}/\texttt{b[i]}]$

   $\equiv \textbf{if } \texttt{k} = \texttt{i} \textbf{ then } \texttt{b[j]} \textbf{ else } \texttt{b[k]} \textbf{ fi} = 0$

   $\Leftrightarrow \textbf{if } \texttt{k} = \texttt{i} \textbf{ then } \texttt{b[j]} = 0 \textbf{ else } \texttt{b[k]} = 0 \textbf{ fi}$ (one possible alternative rewriting)

   $\Leftrightarrow (\texttt{k} = \texttt{i} \rightarrow \texttt{b[j]} = 0) \wedge (\texttt{k} \neq \texttt{i} \rightarrow \texttt{b[k]} = 0)$ (another possible alternative rewriting)

3. $wp(\texttt{b[i]} := \texttt{b[j]},\ \texttt{b[j]} = \texttt{z}) \equiv (\texttt{b[j]} = \texttt{z})[\texttt{b[j]}/\texttt{b[i]}]$

   $\equiv \textbf{if } \texttt{j} = \texttt{i} \textbf{ then } \texttt{b[j]} \textbf{ else } \texttt{b[j]} \textbf{ fi} = \texttt{z}$

   $\Leftrightarrow \texttt{b[j]} = \texttt{z}$

4. $wp(\texttt{b[i]} := 1,\ \texttt{b[i]} = \texttt{b[j]}) \equiv (\texttt{b[i]} = \texttt{b[j]})[1/\texttt{b[i]}]$

   $\equiv (\texttt{b[i]})[1/\texttt{b[i]}] = (\texttt{b[j]})[1/\texttt{b[i]}]$

   $\equiv 1 = (\textbf{if } \texttt{j} = \texttt{i} \textbf{ then } 1 \textbf{ else } \texttt{b[j]} \textbf{ fi})$

   $\Leftrightarrow \texttt{i} = \texttt{j} \vee \texttt{b[j]} = 1$ ($\Leftrightarrow \texttt{i} \neq \texttt{j} \rightarrow \texttt{b[j]} = 1$ if you prefer $\rightarrow$ to $\vee$)

5. $wp(\texttt{b[i]} := \texttt{x};\ \texttt{b[j]} := \texttt{y},\ \texttt{b[i]} \neq \texttt{b[j]})$

   $\equiv wp(\texttt{b[i]} := \texttt{x},\ wp(\texttt{b[j]} := \texttt{y},\ \texttt{b[i]} \neq \texttt{b[j]}))$

   For the embedded wp, $wp(\texttt{b[j]} := \texttt{y},\ \texttt{b[i]} \neq \texttt{b[j]})$

   $\equiv (\texttt{b[i]} \neq \texttt{b[j]})[\texttt{y}/\texttt{b[j]}]$

   $\equiv \texttt{b[i]}\,[\texttt{y}/\texttt{b[j]}] \neq \texttt{b[j]}\,[\texttt{y}/\texttt{b[j]}]$

   $\equiv \textbf{if } \texttt{i} = \texttt{j} \textbf{ then } \texttt{y} \textbf{ else } \texttt{b[i]} \textbf{ fi} \neq \texttt{y}$

   $\Leftrightarrow \texttt{i} \neq \texttt{j} \wedge \texttt{b[i]} \neq \texttt{y}$

   So $wp(\texttt{b[i]} := \texttt{x},\ wp(\texttt{b[j]} := \texttt{y},\ \texttt{b[i]} \neq \texttt{b[j]}))$

   $\Leftrightarrow wp(\texttt{b[i]} := \texttt{x},\ \texttt{i}\neq\texttt{j} \wedge \texttt{b[i]} \neq \texttt{y})$

   $\equiv (\texttt{i} \neq \texttt{j} \wedge \texttt{b[i]} \neq \texttt{y})[\texttt{x}/\texttt{b[i]}]$

   $\equiv \texttt{i} \neq \texttt{j} \wedge \texttt{b[i]}\,[\texttt{x}/\texttt{b[i]}] \neq \texttt{y}$

   $\equiv \texttt{i} \neq \texttt{j} \wedge \texttt{x} \neq \texttt{y}$

   Intuitively, if $\texttt{i} = \texttt{j}$, then we can't have $\texttt{b[i]} \neq \texttt{b[j]}$. Even if $\texttt{i} \neq \texttt{j}$, we need $\texttt{x} \neq \texttt{y}$ to ensure that the assignments $\texttt{b[i]} := \texttt{x};\ \texttt{b[j]} := \texttt{y}$ make $\texttt{b[i]} \neq \texttt{b[j]}$.

6. For $wp(\texttt{b[i]} := \texttt{x},\ \texttt{b[b[i]]} \neq \texttt{b[i]})$, let's first look at substituting $\texttt{x}$ for $\texttt{b[i]}$ in $\texttt{b[b[i]]}$. It's complicated because we have to recursively substitute in the index of the outer $\texttt{b[}\ldots\texttt{]}$. The general rule is

   $(\texttt{b}[e_2])[e_1/\texttt{b}[e_0]] \equiv (\textbf{if } e_2' = e_0 \textbf{ then } e_1 \textbf{ else } \texttt{b}[e_2'] \textbf{ fi})$ where $e_2' \equiv (e_2)[e_1/\texttt{b}[e_0]]$

   So let $e' \equiv (\ \texttt{b[i]}\ )[\texttt{x}/\texttt{b[i]}] \equiv \texttt{x}$, then $(\ \texttt{b[b[i]]}\ )[\texttt{x}/\texttt{b[i]}]$

   $\equiv \textbf{if } e' = \texttt{i} \textbf{ then } \texttt{x} \textbf{ else } \texttt{b}[e'] \textbf{ fi}$

   $\equiv \textbf{if } \texttt{x} = \texttt{i} \textbf{ then } \texttt{x} \textbf{ else } \texttt{b[x]} \textbf{ fi}$

   Then $wp(\texttt{b[i]} := \texttt{x},\ \texttt{b[b[i]]} \neq \texttt{b[i]})$

   $\equiv (\ \texttt{b[b[i]]} \neq \texttt{b[i]}\ )[\texttt{x}/\texttt{b[i]}]$

   $\equiv (\ \texttt{b[b[i]]}\ )[\texttt{x}/\texttt{b[i]}] \neq (\ \texttt{b[i]}\ )[\texttt{x}/\texttt{b[i]}]$

$\equiv$ **if** x = i **then** x **else** b[x] **fi** $\neq$ x

$\Leftrightarrow$ x $\neq$ i $\wedge$ b[x] $\neq$ x

7. For the triple to be valid, it's sufficient to show that its precondition implies the *wp* of the assignment and postcondition. I.e., i < b[i] < b[j] $\rightarrow$ *wp*(b[b[i]] := b[j], b[i] $\neq$ b[j])

First let's calculate *wp*(b[b[i]] := b[j], b[i] $\neq$ b[j])

$\equiv$ (b[i] $\neq$ b[j])[b[j] /b[b[i]]]

$\equiv$ (b[i])[b[j] /b[b[i]]] $\neq$ (b[j])[b[j] /b[b[i]]]

$\equiv$ **if** i = b[i] **then** b[j] **else** b[i] **fi** $\neq$ **if** j = b[i] **then** b[j] **else** b[j] **fi**

$\Leftrightarrow$ **if** i = b[i] **then** b[j] **else** b[i] **fi** $\neq$ b[j]

$\Leftrightarrow$ **if** i = b[i] **then** b[j] $\neq$ b[j] **else** b[i] $\neq$ b[j] **fi**

$\Leftrightarrow$ i $\neq$ b[i] $\wedge$ b[i] $\neq$ b[j]

Going back to our original question, if the implication below is valid, then our triple is valid (because we have the precondition of the triple implying the *wp* of its body and postcondition).

i < b[i] < b[j] $\rightarrow$ *wp*(b[b[i]] := b[j], b[i] $\neq$ b[j])

$\Leftrightarrow$ i < b[i] < b[j] $\rightarrow$ i $\neq$ b[i] $\wedge$ b[i] $\neq$ b[j]

$\Leftrightarrow$ T

So our original triple is indeed valid.

8. swapped(b_1, b_2, i, j) $\equiv$ b_1[i] = b_2[j] $\wedge$ b_1[j] = b_2[i] $\wedge$ ($\forall$ k. (k $\neq$ i $\wedge$ k $\neq$ j $\rightarrow$ b_1[k] = b_2[k]))

9. permuted(b_1, b_2) $\equiv$ b_1 = b_2 $\vee$ ($\exists$ i. $\exists$ j. $\exists$ b_3. swapped(b_1, b_3, i, j) $\wedge$ permuted(b_2, b_3))