Developing algorithms on arrays

We develop several important algorithms on arrays. With each, specify the algorithm by giving its precondition and postcondition as pictures.

Then, draw the invariant by drawing another picture that "generalizes" the precondition and postcondition, since the invariant is true at the beginning and at the end.

Four loopy questions — memorize them:
1. How does loop start (how to make the invariant true)?
2. When does it stop (when is the postcondition true)?
3. How does repetend make progress toward termination?
4. How does repetend keep the invariant true?

Generalize: To derive or induce (a general conception or principle) from particulars.
To make general: render applicable to a wider class

Generalization: All dogs hate cats

square

sides: equal

angles: equal

rhombus

sides: equal

rhombus is a generalization of square
square is a particular kind of rhombus

problem: Tile an 8 x 8 kitchen

generalization: Tile a $2^n \times 2^n$ kitchen (all using L-shaped tiles)

generalization: Tile an $n \times n$ kitchen


Invariant as picture: Generalizing pre- and post-condition
Finding the minimum of an array. Given array $b$ satisfying precondition $P$, store a value in $x$ to truthify postcondition $Q$:

$P$: $b$ with $x$ is the min of this segment

$Q$: $x$ is the min of this segment

The invariant as picture: Generalizing pre- and post-condition
Put negative values before nonnegative ones. Given is precondition $P$:

$P$: $b$ with $x$ is the min of this segment

$Q$: $x$ is the min of this segment

Horizontal notation for arrays, strings, Vectors

Example of an assertion about an array $b$. It asserts that:
1. $b[0..k–1]$ is sorted (i.e., its values are in ascending order)
2. Everything in $b[0..k–1]$ is $\leq$ everything in $b[k..b.length–1]$

Given the index $h$ of the First element of a segment and the index $k$ of the element that Follows the segment, the number of values in the segment is $k – h$.

$b[h..k–1]$ has $k – h$ elements in it.
The invariant as picture: Generalizing pre- and post-condition

Dutch national flag. Swap values of 0..n-1 to put the reds first, then the whites, then the blues. That is, given precondition P, swap value of b[0..n] to truthify postcondition Q:

<table>
<thead>
<tr>
<th>P: b</th>
<th>0</th>
<th>?</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q: b</td>
<td>reds</td>
<td>whites</td>
<td>blues</td>
</tr>
</tbody>
</table>

(values in 0..n-1 are unknown)

???

How to make invariant look like initial condition

<table>
<thead>
<tr>
<th>P: b</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q: b</td>
<td>?</td>
</tr>
</tbody>
</table>

1. Make red, white, blue section empty: use formulas for no. of values in these sections, set j, k, l so that they have 0 elements.

2. Compare precondition with invariant. E.g. in precondition, 0 marks first unknown. In invariant, k marks first unknown. Therefore, k and 0 must be the same.

Partition algorithm: Given an array b[h..k] with some value x in b[h]:

P: b

Swap elements of b[h..k] and store in j to truthify P:

Q: b

change: b

into b

or b

x is called the pivot value.

x is not a program variable; x just denotes the value initially in b[h].

Linear search

Vague spec. Find first occurrence of v in b[h..k-1].

Better spec. Store an integer i to truthify postcondition Q:

Q: b

1. v is not in b[h..i-1]
2. i = k OR v = b[k]

Binary search: Vague spec. Look for v in sorted array segment b[h..k].

Better spec. Store an integer i to truthify:

Postcondition Q: b[h..i] <= v AND v < b[i+1..k]

Below, the array is in non-descending order:

<table>
<thead>
<tr>
<th>P: b</th>
<th>h</th>
<th>?</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q: b</td>
<td>&lt;= v</td>
<td>=&gt; v</td>
<td></td>
</tr>
</tbody>
</table>

Called binary search because each iteration of the loop cuts the array segment still to be processed in half

Reversal: Reverse the elements of array segment b[h..k].

<table>
<thead>
<tr>
<th>P: b</th>
<th>h</th>
<th>not reversed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q: b</td>
<td>reversed</td>
<td></td>
</tr>
</tbody>
</table>

Called binary search because each iteration of the loop cuts the array segment still to be processed in half