Horizontal Notation for Sequences

Example of an assertion about a sequence 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 <= everything in b[k..len(b)-1]

Given index h of the first element of a segment and index k of the element that follows that segment, the number of values in the segment is k - h.

b[h..k-1] has k - h elements in it.

(h+1) - h = 1

Developing Algorithms on Sequences

- Specify the algorithm by giving its precondition and postcondition as pictures.
- Draw the invariant by drawing another picture that "generalizes" the precondition and postcondition
  - The invariant is true at the beginning and at the end
- The four loop design questions
  1. How does loop start (how to make the invariant true)?
  2. How does it stop (is the postcondition true)?
  3. How does the body make progress toward termination?
  4. How does the body keep the invariant true?

Generalizing Pre- and Postconditions

- Dutch national flag: tri-color
  - Sequence of 0..n-1 of red, white, blue "pixels"
  - Arrange to put reds first, then whites, then blues

<table>
<thead>
<tr>
<th>Pre</th>
<th>Post</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>red</td>
<td>&lt; 0</td>
</tr>
<tr>
<td></td>
<td>white</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td></td>
<td>blue</td>
<td>= 0</td>
</tr>
</tbody>
</table>

Put negative values before nonnegative ones.

Finding the minimum of a sequence.

Generalizing Pre- and Postconditions

<table>
<thead>
<tr>
<th>Pre</th>
<th>Post</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt; 0</td>
<td>&gt; 0</td>
</tr>
<tr>
<td></td>
<td>k</td>
<td>j</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

Partitions Algorithm

- Given a sequence b[h..k] with some value x in b[h]:
  - Swap elements of b[h..k] and store in j to truthify post:

```
<table>
<thead>
<tr>
<th>Pre</th>
<th>Post</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>&lt;x</td>
<td>k</td>
</tr>
<tr>
<td></td>
<td>i+1</td>
<td>&gt;= x</td>
</tr>
</tbody>
</table>
```

x is called the pivot value
x is not a program variable
* denotes value initially in b[h]
**Partition Algorithm Implementation**

```python
def partition(b, h, k):
    """Partition list b[h:k] around a pivot x = b[h]""
    i = h; j = k; x = b[h]
    # invariant: b[h..i-1] < x, b[i..j] = x, b[j+1..k] >= x
    while i < j:
        # invariant: b[h..i-1] < x, b[i..j] = x, b[j+1..k] >= x
        if b[i] <= x:
            # Move to end of block.
            _swap(b,i,i+1)
            i = i+1
        else:
            # Move to end of block.
            _swap(b,i+1,j)
            j = j-1
        if post(b,h..i) < x, b[i] is x, and b[i+1..k] >= x
        return i
```

**Dutch National Flag Variant**

- Sequence of integer values
  - 'red' = negatives, 'white' = 0, 'blue' = positive
  - Only rearrange part of the list, not all

**Dutch National Flag Algorithm**

```python
def dnf(b, h, k):
    """Returns partition points as a tuple (i,j)""
    t = h; i = k+1; j = k
    # invar: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        # invar: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
        if b[t] <= 0:
            swap(b,t,t+1)
            t = t+1
        else:
            swap(b,t+1,t)
            i = i+1
        # invar: swap(b,t+1,t)
        # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
        return (i, j)
```

"""Returns partition points as a tuple (i,j)"""

```python
def dnf(b, h, k):
    t = h; i = k+1; j = k
    # invar: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[t] <= 0:
            swap(b,t,t+1)
            t = t+1
        else:
            swap(b,t+1,t)
            i = i+1
        return (i, j)
```