Lecture 25

Designing Sequence Algorithms
Announcements for This Lecture

Assignment & Lab

- A6 is not graded yet
  - Done early next week
  - Survey still open today
- A7 due **Tues, Dec. 4**
  - Some extensions possible
  - But only for major conflicts
- Lab Today: Office Hours
  - Get help on A7 aliens
  - Anyone can go to any lab

Next Week

- Last week of new material
  - Finish sorting algorithms
- The last required lab
  - Material from today, Tues
  - Turn in during consulting
- Week after that is special
  - Last lecture about CS overall
  - Will also have exam details

11/20/18
Horizontal Notation for Sequences

Example of an assertion about an 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 \( \leq \) everything in \( b[k..\text{len}(b)-1] \)

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.
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: b</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>reds</td>
</tr>
<tr>
<td>inv: b</td>
<td>reds</td>
</tr>
</tbody>
</table>
```

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

Make the red, white, blue sections initially empty:
• Range i..i-1 has 0 elements
• Main reason for this trick

Changing loop variables turns invariant into postcondition.
Generalizing Pre- and Postconditions

• Finding the minimum of a sequence.

pre: \( b \) ? \( n \) and \( n \geq 0 \) (values in \( 0..n \) are unknown)

post: \( b \) x is the min of this segment

• Put negative values before nonnegative ones.

pre: \( b \) ? \( n \) and \( n \geq 0 \) (values in \( 0..n \) are unknown)

post: \( b \) < 0 >= 0
Generalizing Pre- and Postconditions

• Finding the minimum of a sequence.

<table>
<thead>
<tr>
<th>pre: b</th>
<th>?</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>n</td>
</tr>
</tbody>
</table>

and \( n \geq 0 \)

(values in 0..n are unknown)

<table>
<thead>
<tr>
<th>post: b</th>
<th>x is the min of this segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>k</td>
<td>j</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inv: b</th>
<th>x is min of this segment</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

(values in j..n are unknown)

• Put negative values before nonnegative ones.

<table>
<thead>
<tr>
<th>pre: b</th>
<th>?</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>n</td>
</tr>
</tbody>
</table>

and \( n \geq 0 \)

(values in 0..n are unknown)

<table>
<thead>
<tr>
<th>post: b</th>
<th>&lt; 0</th>
<th>( \geq 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>n</td>
</tr>
</tbody>
</table>
Generalizing Pre- and Postconditions

- Finding the minimum of a sequence.

  0
  \[\text{pre: } b \begin{array}{l} ? \end{array} \quad \text{and } n \geq 0\]
  0
  \[\text{post: } b \begin{array}{l} \text{x is the min of this segment} \end{array} \]
  0
  \[\text{inv: } b \begin{array}{l} \text{x is min of this segment} \end{array} ? \]

- Put negative values before nonnegative ones.

  0
  \[\text{pre: } b \begin{array}{l} ? \end{array} \quad \text{and } n \geq 0\]
  0
  \[\text{post: } b \begin{array}{l} < 0 \end{array} \geq 0 \]

(values in 0..n are unknown)

(values in j..n are unknown)
Generalizing Pre- and Postconditions

• Finding the minimum of a sequence.

<table>
<thead>
<tr>
<th>pre: b</th>
<th>?</th>
<th>and n &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>x is the min of this segment</td>
<td>n</td>
</tr>
<tr>
<td>inv: b</td>
<td>x is min of this segment</td>
<td>?</td>
</tr>
</tbody>
</table>

• Put negative values before nonnegative ones.

<table>
<thead>
<tr>
<th>pre: b</th>
<th>?</th>
<th>and n &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>&lt; 0</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>inv: b</td>
<td>&lt; 0</td>
<td>?</td>
</tr>
</tbody>
</table>
Generalizing Pre- and Postconditions

• Finding the minimum of a sequence.

<table>
<thead>
<tr>
<th>pre:  b</th>
<th>?</th>
<th>and n &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>x is the min of this segment</td>
<td></td>
</tr>
<tr>
<td>inv:   b</td>
<td>x is min of this segment</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pre:  b</th>
<th>?</th>
<th>and n &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>x is the min of this segment</td>
<td></td>
</tr>
<tr>
<td>inv:   b</td>
<td>x is min of this segment</td>
<td>?</td>
</tr>
</tbody>
</table>

• Put negative values before nonnegative ones.

<table>
<thead>
<tr>
<th>pre:  b</th>
<th>?</th>
<th>and n &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>&lt; 0</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>inv:   b</td>
<td>&lt; 0</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pre:  b</th>
<th>?</th>
<th>and n &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>&lt; 0</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>inv:   b</td>
<td>&lt; 0</td>
<td>?</td>
</tr>
</tbody>
</table>
Partition Algorithm

• Given a sequence $b[h..k]$ with some value $x$ in $b[h]$:

  \[
  \begin{array}{c}
  \text{pre:} \\
  \text{b} & \begin{array}{c}
  \hline
  x \\
  \end{array}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{h} & \begin{array}{c}
  \hline
  \end{array} & \begin{array}{c}
  k
  \end{array}
  \end{array}
  \]

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

  \[
  \begin{array}{c}
  \text{post:} \\
  \begin{array}{c}
  \hline
  \leq x & x & \geq x
  \end{array}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{b} & \begin{array}{c}
  \hline
  \end{array}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{h} & \begin{array}{c}
  \hline
  \end{array} & \begin{array}{c}
  i & i+1 & \begin{array}{c}
  \hline
  \end{array} & \begin{array}{c}
  k
  \end{array}
  \end{array}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{change:} \\
  \begin{array}{c}
  \hline
  3 & 5 & 4 & 1 & 6 & 2 & 3 & 8 & 1
  \end{array}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{into} \\
  \begin{array}{c}
  \hline
  1 & 2 & 1 & 3 & 5 & 4 & 6 & 3 & 8
  \end{array}
  \end{array}
  \]

  • $x$ is called the pivot value
    - $x$ is not a program variable
    - denotes value initially in $b[h]$
Partition Algorithm

• Given a sequence $b[h..k]$ with some value $x$ in $b[h]$:  

\[
\begin{array}{c}
\text{pre: } b [h..k] \\
\text{post: } b [h..k]
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{h} \\
x
\end{array} & \begin{array}{c}
? \\
\end{array} \\
\begin{array}{c}
h \\
\end{array} & \begin{array}{c}
i & i+1 \\
\end{array} & \begin{array}{c}
k \\
\end{array}
\end{array}
\]

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

\[
\begin{array}{c}
\begin{array}{c}
\text{h} \\
\end{array} & \begin{array}{c}
i \\
\end{array} & \begin{array}{c}
i+1 \\
\end{array} & \begin{array}{c}
k \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{b} \\
3 \ 5 \ 4 \ 1 \ 6 \ 2 \ 3 \ 8 \ 1
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{b} \\
1 \ 2 \ 1 \ 3 \ 5 \ 4 \ 6 \ 3 \ 8
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{b} \\
1 \ 2 \ 3 \ 1 \ 3 \ 4 \ 5 \ 6 \ 8
\end{array}
\end{array}
\]

• $x$ is called the \textbf{pivot value}
  
  ▪ $x$ is not a program variable
  ▪ denotes value initially in $b[h]$
Partition Algorithm

- Given a sequence $b[h..k]$ with some value $x$ in $b[h]$:  
  
  \[
  \begin{array}{c}
  \text{pre: } b \\
  \hline
  h & x & \text{?} & k
  \end{array}
  \]

- Swap elements of $b[h..k]$ and store in $j$ to truthify post:  
  
  \[
  \begin{array}{c}
  \text{post: } b \\
  \hline
  h & i & i+1 & k \\
  \hline
  \leq x & x & \geq x
  \end{array}
  \]
Partition Algorithm

• Given a sequence b[h..k] with some value x in b[h]:

\[
\begin{array}{c|c|c|}
h & x & ? \\
\hline
\text{pre: } b & \text{<= x} & \text{>= x} \\
\end{array}
\]

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

\[
\begin{array}{c|c|c|c|}
h & i & i+1 & k \\
\hline
\text{post: } b & \text{<= x} & \text{x} & \text{>= x} \\
\end{array}
\]

• Agrees with precondition when i = h, j = k+1
• Agrees with postcondition when j = i+1
def partition(b, h, k):
    
    """Partition list b[h..k] around a pivot x = b[h]"""

    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x

    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            _swap(b,i+1,j-1)
            j = j - 1
        else:
            # b[i+1] < x
            _swap(b,i,i+1)
            i = i + 1

    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x

    return i

**partition(b,h,k), not partition(b[h:k+1])**
Remember, slicing always copies the list!

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

<= x  x  ?  >= x
h  i  i+1  j  k
1  2  3  1  5  0  6  3  8
def partition(b, h, k):

    """Partition list b[h..k] around a pivot x = b[h]"""

    i = h; j = k+1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x

    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            _swap(b,i+1,j-1)
            j = j - 1
        else:  # b[i+1] < x
            _swap(b,i,i+1)
            i = i + 1

    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x

    return i
## Partition Algorithm Implementation

```python
def partition(b, h, k):
    
    i = h; j = k + 1; x = b[h]
    # invariant: b[h..i-1] < x, b[i] = x, b[j..k] >= x

    while i < j - 1:
        if b[i + 1] >= x:
            # Move to end of block.
            `_swap(b, i + 1, j - 1)
            j = j - 1
        else:  # b[i+1] < x
            `_swap(b, i, i + 1)
            i = i + 1

    # post: b[h..i-1] < x, b[i] is x, and b[i+1..k] >= x
    return i
```

<table>
<thead>
<tr>
<th>&lt;= x</th>
<th>x</th>
<th>?</th>
<th>&gt;= x</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>i+1</td>
<td>j</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>i+1</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>h</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Dutch National Flag Variant

- Sequence of integer values
  - ‘red’ = negatives, ‘white’ = 0, ‘blues’ = positive
  - Only rearrange part of the list, not all

<table>
<thead>
<tr>
<th>pre: b</th>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>post: b</td>
<td>h</td>
<td>t</td>
</tr>
<tr>
<td>inv: b</td>
<td>h</td>
<td>&lt;0</td>
</tr>
</tbody>
</table>
Dutch National Flag Variant

- Sequence of integer values
  - ‘red’ = negatives, ‘white’ = 0, ‘blues’ = positive
  - Only rearrange part of the list, not all

pre:  

<table>
<thead>
<tr>
<th>h</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

post:  

<table>
<thead>
<tr>
<th>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>=0</td>
<td>&gt;0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

inv:  

<table>
<thead>
<tr>
<th>h</th>
<th>t</th>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>?</td>
<td>=0</td>
<td>&gt;0</td>
<td></td>
</tr>
</tbody>
</table>

pre:  

| t = h, |
| i = k+1, |
| j = k |

post:  

| t = i |
def dnf(b, h, k):
    
    # Returns: partition points as a tuple (i,j)
    t = h; i = k + 1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] > 0, b[i..j] = 0, b[j+1..k] > 0

    while t < i:
        if b[i-1] < 0:
            swap(b, i-1, t)
            t = t + 1
        elif b[i-1] == 0:
            i = i - 1
        else:
            swap(b, i-1, j)
            i = i - 1; j = j - 1

    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
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;
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b,i-1,t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b,i-1,j)
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```

<table>
<thead>
<tr>
<th>&lt; 0</th>
<th>?</th>
<th>= 0</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>t</td>
<td>i</td>
<td>j</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
```
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b,i-1,t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b,i-1,j)
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
def dnf(b, h, k):
    """Returns: partition points as a tuple (i,j)"""
    t = h; i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] ? b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            swap(b,i-1,t)
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            swap(b,i-1,j)
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
Will Finish This Next Week