Complete the following proof outline validly. Keep it as simple as possible.

## $a = b = false$

### Global Inv: __________________________

co

## $a, w := true, 13$

## $\langle await(b) \rangle$

## $x := f(w, y)$

## $//$

## $b, y := true, 42$

## $\langle await(a) \rangle$

## $z := g(w, y)$

oc

## $x = f(13, 42) \land z = g(13, 42)$
Q1 [10] Future. A “future” is an object that either represents a value, or will represent a value in the future. Typically a future is returned from a function that may spin off a new thread to compute the value. For example

```java
Future future = someObj.someMethod();
...
Object value = future.get();
```

The thread getting the value may have to wait until the value has been put. Using Java notation design and implement a Future class. It should support the following methods.

```java
public void put( Object pValue );
public Object get();
```

You may use my Signal-and-Wait monitor package, if you want.
In what ways do remote procedure calls differ from ordinary (local) procedure calls?

Q3 [10] Bag of tasks product. Consider the problem of computing the ‘product’ of all items in an array $a[0..N-1]$.

$$a[0] \otimes a[1] \otimes ... \otimes a[N-1]$$

We can assume that $\otimes$ is some associative operator, which may be time consuming to compute.

For simplicity assume $N = 2^n$ where $n$ is an integer.

We can make use of a tree structure, filling in an array $A[0..n, 0..N-1]$. So that $A[0, j] = a[j]$ and $A[i, j] = A[i, 2 \times j] \otimes A[i, 2 \times j + 1]$, where $0 < i \leq n$ and $0 \leq j \leq 2^{n-1}$. This is illustrated for $N = 8, n = 3$:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$A[1, 0] \otimes A[1, 1]</td>
<td>A[1, 2] \otimes A[1, 3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$A[2, 0] \otimes A[2, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Design a bag of tasks algorithm in which each task consists of filling in one element of $A$.

In case it helps, let’s define the following functions to help navigate the tree

$sibling((i, j)) = (i, j + 1)$, if $j$ is even and $i < n$

$sibling((i, j)) = (i, j - 1)$, if $j$ is odd and $i < n$

$parent((i, j)) = (i + 1, \lfloor j / 2 \rfloor)$, if $i < n$

$children((i, j)) = \{ (i - 1, 2j), (i - 1, 2j + 1) \}$, if $i > 0$

Use the course’s design notation. Feel free to declare any auxiliary data structures you would like.

Hint: Ensure that each ‘parent’ task goes in the bag only after both of its child tasks have
filled in their elements of the A array.