Finding Strong Loop Invariants

Assume there is a function, `int slowFib(int n)`, which follows the following recursive definition:

```
slowFib(0) = 0
slowFib(1) = 1
slowFib(n) = slowFib(n-1) + slowFib(n-2)     (for n ≥2)
```

A naïve implementation of `slowFib` using only this recursive definition would likely take exponential time, so we would like to implement a faster version.
Define the function

```
 1  int[] fib(int n)
 2  //@requires n >= 0;
 3  //@ensures \length(\result) == n+1;
 4  //@ensures \result[n] == slowFib(n);
 5  {
 6    int[] f = alloc_array(int, n+1);
 7    f[0] = 0; // this is safe because n >= 0
 8    if(n >= 1)
 9      f[1] = 1;
10    if(n >= 2) {
11      for(int i = 2; i < n+1; i++)
12      //@loop_invariant 2 <= i && i <= n+1;
13      //@loop_invariant ?
14      {
15        f[i] = f[i-1] + f[i-2];
16      }
17      return f;
18    }
19  }
```

The first loop invariant should come automatically. For any loop that iterates over an array we want to give a loop invariant that ensures the array is accessed "safely," that is, we don't try to get an index that is out of bounds. While we can achieve this by simply having a lower bound on `i` and then using the loop guard, if we want to argue about the value of `i` at the end of the loop, we need more than just the loop guard, so you might as well put in an upper bound for good practice (remember the upper bound is inclusive whereas the lower bound is not).

First, let's prove the first loop invariant. We need to show two things:

1) The loop invariant holds before the start of the loop

2) IF the loop invariant holds at the start of an iteration, then it will also hold at the end

Before the loop:

`i` = 2 (line 11) ≤ `n` (line 10) ⇒ 2 ≤ `i` ≤ `n`+1

Assume the loop invariant holds at the beginning of an iteration (2 ≤ `i` ≤`n`+1).
At the end of the iteration `i'` = `i`+1 (line 11).
WTS 2 ≤ `i'` ≤ `n`+1

2 ≤ `i` (LI) ⇒ 2 ≤ `i`+1 = `i'`
`i` < `n`+1 (line 11) ⇒ `i`+1 = `i'` ≤ `n`+1

(Note that we did not simply say that `i` starts at 2 and is incremented until it reaches `n`+1. This is a hand-wavy argument that assumes we know what the loop is doing. We want to actually PROVE that it does what we think/claim it does.)

At this point we have proved our first loop invariant. Now we want to show that the post-conditions hold.

We have two post-conditions (lines 3 and 4). We will prove both of them.
Line 3: This one is easy. We return `f`, and in line 6 we allocated `f` with `n+1` indices. Since the array size is fixed, `\length (f)` must be `n+1`.

Line 4: This will take some more work. We have 3 cases: either `n = 0`, `n = 1`, or `n >= 2`.
The first two are easy. If `n = 0`, then we set `f[n]` to `0 == slowFib(0)`. If `n = 1`, then we set `f[n]` to `1 == slowFib(1)` .

In the third case, we need loop invariants to prove the post condition. This is where the notion of strong loop invariants comes from. We want loop invariants that are sufficient to prove the post condition holds, not simply observations about what is true during the loop (otherwise we would just write `//@loop_invariant true` and be done, but this is useless).
To come up with the loop invariant, we can think about what we want to know at the end of the loop. We want to prove that `f[n] == slowFib(n)`. First, we notice that at the end of the loop, `i == n+1`.

(NOTE: The reason we know `i == n+1` is because the loop invariant on line 12 tells us that `i <= n+1` and the negation of the loop guard gives us that `i >= n+1`, thus `i` must be `n+1`. DO NOT argue that we incremented `i` until it became equal to `n+1` as this argument is merely stating what we want to prove and assuming the code does what we think it does)

Therefore, it makes sense to consider having `f[i-1] == slowFib(i-1)` as a loop invariant—after all, if we can prove this, then we can easily conclude the post-condition holds by plugging in `n+1` for `i`. Remember, the post-condition should generally follow easily from the loop invariants.
Let's try that. Remember, we have to show the loop invariant holds before the loop and after each iteration:

Before the loop: `i` is 2 `f[i-1] == f[2-1] == f[1] == 1 == slowFib(1)`
At the end of each iteration:
(1) Assume the invariant holds at the beginning of the iteration ie
`f[i-1] == slowFib(i-1).`
(2) First, we set `f[i] = f[i-1] + f[i-2]`, then we increment `i'` to `i+1`.
We now WTS that `f[i'-1] == slowFib(i'-1)`, ie `f[i] == slowFib(i).`
Using our LI's we can substitute `slowFib(i-1)` for `f[i-1]` in `f[i] = f[i-1] + f[i-2].`
to get `f[i] == slowFib(i-1) + f[i-2].`

Unfortunately, we are now stuck because we don't know anything about `f[i-2]`, so... another loop invariant? If we knew `f[i-2]` equaled `slowFib(i-2)` we could use the recursive definition to get `f[i]`. So let's add that as a loop invariant.
Quickly, let's work out the base case (when `i` is 2) for this loop invariant:
`f[i-2] == f[2-2] == f[0] == 0 == slowFib(0).`

Now we can also assume this loop invariant (given we can prove it also holds at the end) so we get
`f[i] == slowFib(i-1) + slowFib(i-2).`

From the definition of `slowFib(n)` when `n` ≥ 2, this gives us that `f[i] == slowFib(i),` which is what we needed to show for the first part of our invariant.

Now all we need to show is that the second part still holds, ie
`f[i'-2] == f[i-1] == slowFib(i'-2) == slowFib(i-1).`
This is actually easy, since we already assumed that `f[i-1] == slowFib(i-1)` in the first part of our invariant.

We have now provided and prove a final loop invariant,
`//@loop_invariant f[i-1] == slowFib(i-1) && f[i-2] == slowFib(i-2)`
Now, we simply claim that since at the end, `i` is `n`+1 (which we proved earlier), at the end of the loop,
`f[n] == f[n+1-1] == slowFib(n+1-1) == slowFib(n).`

Technically, we must also prove termination, which we do by arguing that `i'` must be strictly greater than `i` after each iteration. From the code (line 11) we have that `i' = i+1 > i`. Thus `i` is strictly increasing so it must eventually become at least `n+1`.