CPSC 331 — Solutions for Tutorial Exercise #4

The questions in this exercise concern the algorithm shown below — which is supposed to solve the "Fibonacci Number Computation" problem that has been considered in recent lectures.

```
integer betterFibLoop (integer n) {
// Assertion:  A nonnegative integer n has been given as input.
 1. if (n == 0) {
 2.  return 0
   } else {
 3.  integer oldest := 0
 4.  integer middle := 1
 5.  integer i := 1
 6.  while (i < n) {
 7.    integer youngest := oldest + middle
 8.    oldest := middle
 9.    middle := youngest
10.    i := i + 1
     }
11.  return middle
   }
}
```

1. You were first asked to prove that the following is a ***loop invariant*** for the `while` loop in this algorithm.

   Loop Invariant:

   1. `n` is an integer input such that $\texttt{n} \geq 1$.
   2. `i` is an integer variable such that $1 \leq \texttt{i} \leq \texttt{n}$.
   3. `oldest` is an integer variable with value $F_{\texttt{i}-1}$.
   4. `middle` is an integer variable with value $F_{\texttt{i}}$.

   Solution: **Loop Theorem #1** will be used to prove this.

   (a) The loop test, at line 6 in this algorithm, has no side-effects.

   (b) Consider an execution of this algorithm beginning with the precondition for the "Fibonacci Number Computation" problem satisfied. There is nothing to be proved if the `while` loop is never reached and executed at all — so it suffices to consider

the case that the text at line 1 fails and lines 3–5 and 6 are executed. One can see by the inspection of the code that the loop can never be executed a second time, so it suffices to consider a ***first*** execution of this loop.

The precondition for the "Fibonacci Number Computation" problem implies that `n` is an integer variable such that $\texttt{n} \geq 0$ when the execution of the algorithm begins — and the value of `n` is never changed. Furthermore, `n` cannot be equal to zero if the loop is reached (because the test at line 1 fails) so $\texttt{n} \geq 1$, as required to establish part 1 of the loop invariant when the execution of the loop begins.

One can see by the inspection of the code (noting line 5) that `i` is an integer variable with value 1 when the execution of the loop begins and, since $\texttt{n} \geq 1$, this establishes that part 2 of the loop invariant is satisfied when the execution of the loop begins as well.

Since `oldest` is an integer variable whose value is set (at line 3) to be $0 = F_0 = F_{\texttt{i}-1}$ and `middle` is an integer variable whose value is set (at line 4) to be $1 = F_1 = F_{\texttt{i}}$, parts 3 and 4 are satisfied when the execution of the loop begins too.

Thus the proposed "loop invariant"' is satisfied when the execution of the loop begins.

(c) Consider an execution of the *body* of this loop that begins with the proposed "loop invariant" satisfied.

It follows by part 1 of the loop invariant that `n` is (initially) an integer variable such that $\texttt{n} \geq 1$ and, since the value of `n` is not changed during executions of lines 7–10, this is also true when this execution of the loop ends.

It follows by part 2 of the loop invariant that `i` is an integer variable such that $1 \leq \texttt{i} \leq \texttt{n}$. Since the test at line at line 6 was checked and passed, $\texttt{i} < \texttt{n}$, so that (since `i` and `n` both have integer values) $1 \leq \texttt{i} \leq \texttt{n} - 1$ at this point. While the value of `n` is never changed, the value of `i` is increased by one during the execution of line 10 — so that $2 \leq \texttt{i} \leq \texttt{n}$ at the end of this execution of the loop body, and part 2 of the proposed "loop" invariant is satisfied at the end of this execution of the loop body as well.

It follows by parts 3 and 4 that `oldest` and `middle` are integer variables such that $\texttt{oldest} = F_{\texttt{i}-1}$ and $\texttt{middle} = F_{\texttt{i}}$ at the beginning of this execution of the loop body. Since $\texttt{i} \geq 1$, $\texttt{i} + 1 \geq 2$ and it follows that $\texttt{youngest} = F_{\texttt{i}-1} + F_{\texttt{i}} = F_{\texttt{i}+1}$ immediately after the execution of line 7. Thus $\texttt{oldest} = F_{\texttt{i}}$ and $\texttt{middle} = F_{\texttt{i}+1}$ immediately after the step at line 9. However, since `i` is incremented at line 0, $\texttt{oldest} = F_{\texttt{i}-1}$ and $\texttt{middle} = F_{\texttt{i}}$ again at the end of this execution of this loop body, re-establishing parts 3 and 4 once again.

Thus the proposed "loop invariant" is satisfied, once again, when this execution of the loop body ends.

It follows by "Loop Theorem #1" that this is a loop invariant for this loop, as claimed.

2. You were next asked to use this to prove that the `betterFibLoop` algorithm is ***partially correct***.

   Solution: One can see by an inspection of the signature and instructions in this algorithm that it has no undocumented side-effects — it does not access any inputs (or global data), change input values, modify global data, or create outputs unless this is documented in the specification of requirements for the "Fibonacci Number Computation" problem.

   It there suffices to show that if this algorithm is executed, when the precondition for the "Fibonacci Number Computation" problem is satisfied, then either

   (a) the execution of the algorithm halts, and the postcondition for the "Fibonacci Number Computation" problem is satisfied when this happens, or
   (b) the execution of the algorithm never halts at all.

   With that noted, consider an execution of this algorithm when the precondition of this problem is initially satisfied — so that $\mathtt{n}$ is an integer input such that $\mathtt{n} \geq 0$.

   - Suppose, first, that $\mathtt{n} = 0$. In this case the test at line 1 is passed and the execution of the algorithm halts after the execution of the step at line 2 — with $0 = F_0 = F_{\mathtt{n}}$ being returned, as needed to satisfy condition (a).
   - The only other case is that $\mathtt{n} \geq 1$, so that the test at line 1 fails, the steps at line 3–5 are executed, and the `while` loop is reached and executed after that. If the execution of the loop never ends then the execution of the algorithm never ends, either, and condition (b) is satisfied.

     Otherwise the execution of the loop ends — and the above ***loop invariant*** is satisfied at this point, so that $\mathtt{i}$ is an integer variable and $\mathtt{n}$ is an integer input such that $1 \leq \mathtt{i} \leq \mathtt{n}$, by parts 1 and 2. Since the test at line 6 much have been checked and ***failed***, in order for the execution of the loop to end, $\mathtt{i} \geq \mathtt{n}$ as well — so that $\mathtt{i} = \mathtt{n}$ at the end of the execution of the `while` loop.

     It now follows by part 4 of the loop invariant that $\mathtt{middle} = F_{\mathtt{i}} = F_{\mathtt{n}}$, so that the $\mathtt{n}^{\text{th}}$ Fibonacci number has been returned as output after the execution of the step at line 11: Condition (a) has been satisfied in this case.

   Since either condition (a) or condition (b) is satisfied in every possible case it follows that this algorithm is ***partially correct***, as claimed.

3. You were next asked to give a ***bound function*** for the `while` loop in your algorithm and to show that your answer is correct.

   Solution: The function $f(\mathtt{n}, \mathtt{i}) = \mathtt{n} - \mathtt{i}$ is a ***bound function*** for the `while` loop in this algorithm.

   To see that this is the case, note the following:

- Since `n` is an integer input, and `i` is an integer variable — whose value is defined before the `while` loop is reached — this is certainly a well-defined total function of the inputs and local variables of this algorithm.
- During an execution of the loop body (that is part of an execution of the algorithm when the precondition for the "Fibonacci Number Computation" problem is initially satisfied) the value of `n` is not changed but the value of `i` is increased by one, as a result of the execution of the step at line 10. Thus the value of the function f is ***decreased*** by (at least) one whenever the body of the `while` loop is executed.
- Suppose that the value of this function is less than or equal to zero, that is, $\texttt{n} - \texttt{i} \leq 0$. Then $\texttt{i} \geq \texttt{n}$ and the loop test at line 6 fails — causing the execution of the loop to halt.

Thus the above function is a ***bound function*** for the `while` loop in this algorithm, because it satisfies all the properties of one.

4. You were next asked to use this to show that every execution of this algorithm, that starts with the precondition for the "Fibonacci Number Computation" problem satisfied, ***terminates***.

   Solution: Consider an execution of this algorithm beginning when the precondition for the "Fibonacci Number Computation" problem is satisfied — so that `n` is an integer input such that $\texttt{n} \geq 0$.

   If $\texttt{n} = 0$ then the test at line 1 passes and the execution of the algorithm halts after the execution of the step at line 2.

   Otherwise $\texttt{n} \geq 1$, the test at line 1 fails, the steps at lines 3–5 are executed, and and an execution of this `while` loop begins.

   - The loop test at line 6 is a simple comparison of an integer input and variable — so it has no side-effects, and every execution of this loop halts.
   - The body of the `while` loop is simply a sequence of four assignment statements, so every execution of the body of the `while` loop halts too.
   - As noted above, this `while` loop has a bound function.

   It now follows by "Loop Theorem #2" that this execution of the `while` loop terminates. The execution of the algorithm then terminates after the execution of the step at line 11.

   this establishes the claim. As noted in the tutorial exercise, it now follows that the `betterFibLoop` algorithm is ***correct***.