Answering the four loopy questions

We now answer the four loopy questions concerning the algorithm given at the end of this document, along with the flow chart.

1. First loopy question. Does the algorithm start right: does it make invariant P true? Since the initialization sets z to 0 and k to m, replace z and k in invariant P by 0 and m, giving

m <= m <= n+1 and 0 = sum of m..m–1

and show that this is true. Precondition Q tells us that m ≤ n+1. Second, the sum of the empty range m..m-1 is 0. Therefore, we see that this assertion is true. Hence, the algorithm starts right.

2. Second loopy question. Does k > n && P => R? To see this, work as follows. The invariant P requires that k ≤ n+1. Since we also know that k > n, we can determine that k = n+1. Since k = n+1, replace k by n+1 in P, giving:

m <= n+1 <= n+1 and z = sum of m..n+1–1

The last term reduces to z = sum of m..n, which is R. Hence, the answer to the second loopy question is: yes.

3. Third loopy question. Does the repetend make progress toward termination? Whenever the repetend is executed, k ≤ n and the iteration increases n by 1. Hence, at some point, k equals n+1 and the loop terminates. So the answer to the third loopy question is: yes.

4. Fourth loopy question. Does the repetend keep P true, i.e. is {k <= n && P} z= z+k; k= k+1; {P} true? First, you can see that the first part m ≤ k ≤ n+1 is maintained by the repetend. Second, by the invariant, z = sum of m..k-1. After executing z= z+k; z = sum of m..k. Then, after k= k+1; again z = sum of m..k-1. Thus, the answer to the fourth loopy question is: yes.

Summary. The answer to all four loopy questions is: yes. Hence, the loop (with initialization) is correct.

Discussion. It took a lot of words to prove that the loop was correct. If you want to explain the correctness in detail to someone, you have to do this. But, as you become experienced with doing this, you will check the loopy questions in your mind quite quickly, without having to actually say all this or write it down. Your mind will go through each point, quite quickly. With experience, you will find this is *far less work* and *less error prone* than working without the invariant and trying to understand the algorithm by hand-executing it on a few test cases.

```
// Store the sum of m..n in z
// precondition Q: m <= n+1
int z= 0;
int k= m;
// invariant P: m <= k <= n+1 and z = sum of m..k–1
while (k <= n) {
   z= z + k;
   k= k + 1;
}
// postcondition R:  z = sum of m..n
```