The loop invariant and four loopy questions

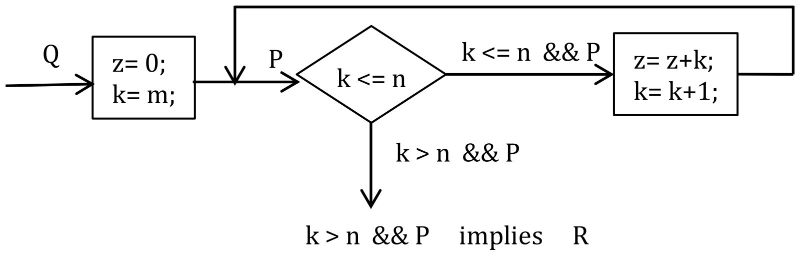
This algorithm stores the sum of m..n in variable z. Execute it by hand with m = 5 and n = 7; you will see that it stores 18 in z. But looking at one test case is not a convincing argument that the algorithm is correct.

```
// 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
```

Precondition Q is simply the restriction on ranges m..n —m can't be bigger than n+1. Postcondition R states that the algorithm indeed stores the sum of m..n in z, that it did the task required by the comment on the first line.

Just before the while-loop there is a comment, giving an "invariant" P. We explain what this means.

Consider the following flow chart. Giving an invariant just before the loop means that the assertions on each edge of the flow chart are true. Notice that P is to be true on every edge, once the initialization makes P true. "Invariant" means non-changing. Once P is true, its value doesn't change; it is invariant. Since P is to be true before and after each iteration (each execution of the repetend), its value doesn't change.



There are three pieces of code: the initialization, the loop condition, and the repetend. We look carefully at each one, coming up with *four loopy questions* that are inspired by this flow chart. We will answer the questions later.

1. **Does it start right: Does the initialization make P true?** Look at the beginning of the flowchart. Given Q true, execution of the initialization is supposed truthify P. We can write this succinctly as: Does {Q} z= 0; k= m; {P} hold?

2. **Does it stop right: is postcondition R true when the loop terminates?** Based on the flowchart, when k > n is true, the loop terminates. At that time, the loop condition is false and invariant P is true. That's *all* we know. Therefore, from these two facts, we need to conclude that postcondition R is true. Thus, the second loopy question is: Does it stop right: Does k > n together with invariant P imply R?

3. **Does each iteration (each execution of the repetend) make progress toward termination?** The repetend must do something to get closer to making the loop condition false. Thus, the third loop question is: Does the repetend make progress toward termination?

4. **Does the repetend keep invariant P true?** According to the flow chart, P is true before the loop condition, and if the loop condition is true, the repetend is executed. The flow chart indicates that the invariant must be true after the repetend is executed. So, the fourth loopy question is: Does the repetend keep P true? We state this more formally like this: Is {k ≤ n && P} z= k; k= k+1; {P} true?

If the answer to these four loopy questions is correct, then we *know* that the loop with initialization is correct. We have *proved* correctness.

All loops can be proved correct in this fashion. Given a suitable invariant, we just have to answer these four loopy questions. In the next video, we show how to check the four loopy questions for this little algorithm.