# Loop Invariants and Proofs of Correctness

Ekkapot Charoenwanit
Efficient Algorithms

September 5, 2020

## 1 Background

A loop invariant is a proposition that is always true at the start of each iteration of a loop. A sufficiently strong loop invariant should imply the post-condition of the algorithm at the end of the last iteration so that we can use it to infer whether the result produced on termination is correct. We use loop invariants as a tool to understand and prove the correctness of algorithms. To prove that a loop invariant holds, we must show that the first two of the following three properties hold:

**Initialization**: The invariant is true prior to the start of the first iteration.

**Maintenance**: Assuming the invariant is true at the start of an iteration, it remains true at the start of the next iteration.

To show that the algorithm produces the correct result, we apply the loop invariant, which we have already proved true, to the point where the loop has just exited. Thereby, a useful property should immediately follow, which can help show the correctness of the algorithm.

**Termination**: When the loop exits, the loop invariant provides useful information that helps demonstrate that the algorithm is correct.

## 2 Examples

**Example 1.** *Show that the following program correctly computes the sum of all the $n$ elements in the array $A[1...n]$.*

**Algorithm 1** implements $\sum_{i=1}^{n} A[i]$

```
1: procedure SUM($A[1...n]$)
2:     $v \leftarrow 0$
3:     for $i = 1 \rightarrow n$ do
4:         $v \leftarrow v + A[i]$
5:     return $v$
```

**Proof**: We will show the correctness of the program using loop invariant analysis.

**Loop Invariant**: At the start of each iteration $i$ of the for loop, $v$ stores the sum of the values of the elements in the subarray $A[1...i-1]$.

**Initialization**: Prior to the first iteration, we have $i = 1$.
By code inspection, at the start of the first iteration, $v = 0$ as a result of the assignment in Line 2.
By the loop invariant, at the start of the first iteration, $v$ stores the sum of the values of the elements in the

subarray $A[1...i-1] \equiv A[1...1-1] \equiv A[1...0]$, which is an empty array.Therefore, the sum is (trivially) 0. Hence, the loop invariant agrees with what the code does. The loop invariant then holds at the start of the first iteration.

**Maintenance**: Assuming the invariant holds at any iteration $i$, we must show that the loop invariant remains true at the start of the next iteration $i^{'}$. In other words, we must show that the following proposition holds:

**At the start of the iteration $i^{'}$ of the for loop, $v$ stores the sum of the values of the elements in the subarray $A[1...i^{'}-1]$.**

Let us consider an arbitrary iteration $i$. At the start of the iteration $i$, $v$ is the sum of the values of the elements in $A[1...i-1]$. During the execution of the iteration $i$, $v$ is incremented by $A[i]$. Therefore, $v$ now stores the sum of the values of the elements in $A[1...i]$.

At the end of the iteration $i$, $i$ is incremented by one. Therefore, we have $i^{'} = i+1$.
Therefore, at the start of the iteration $i^{'}$, $v$ is the sum of the values of the elements in $A[1...i] \equiv A[1...i^{'}-1]$, thereby reestablishing the loop invariant as required.

**Termination**: When the loop exits, we have $i = n+1$. Substituting $i = n+1$ into the loop invariant, we have that $v$ stores the sum of all the values of the elements in $A[1...i-1] \equiv A[1...(n+1)-1] \equiv A[1...n]$, which is the entire array. $\square$

**Example 2.** *Show that the following program correctly computes the factorial of $n$ assuming $n \geq 0$.*

**Algorithm 2** implements the factorial of $n$

```
procedure FACTORIAL($n$)
    $v \leftarrow 1$
    for $i = 1 \rightarrow n$ do
        $v \leftarrow v * i$
    return $v$
```

**Proof**: We will show the correctness of the program using loop invariant analysis.

**Loop Invariant**: At the start of each iteration $i$ of the for loop, $v = (i-1)!$.

**Initialization**: Prior to the first iteration, we have $i = 1$.
By code inspection, at the start of the first iteration, $v = 1$ as a result of the assignment in Line 2.
By the loop invariant, at the start of the first iteration, $v$ stores $(i-1)! = (1-1)! = 0! = 1$.
Hence, the loop invariant agrees with what the code does. The loop invariant then holds at the start of the first iteration.

**Maintenance**: Assuming the invariant holds at any iteration $i$, we must show that the loop invariant remains true at the start of the next iteration $i^{'}$. In other words, we must show that the following proposition holds:

**At the start of the iteration $i^{'}$ of the for loop, $v = (i^{'}-1)!$.**

Let us consider an arbitrary iteration $i$. At the start of the iteration $i$, $v = (i-1)!$. During the execution of the iteration $i$, $v$ is updated to $(i-1)! \cdot i$, which evaluates to $i!$, in Line 4. Therefore, we currently have $v = i!$.

At the end of the iteration $i$, $i$ is incremented by one. Therefore, we have $i^{'} = i+1$.
Therefore, at the start of the iteration $i^{'}$. $v = i! = (i^{'}-1)!$, thereby reestablishing the loop invariant as

required.

**Termination**: When the loop exits, we have $i = n + 1$. Substituting $i = n + 1$ into the loop invariant, we have $v = (i - 1)! = (n + 1 - 1)! = n!$, which is what the factorial program must produce on termination. $\square$

**Example 3.** *Show that the following program correctly computes the factorial of $n$ assuming $n \geq 0$.*

**Algorithm 3** implements the factorial of $n$

```
procedure FACTORIAL($n$)
    $r \leftarrow 1$
    $k \leftarrow n$
    while $k > 0$ do
        $r \leftarrow k * r$
        $k \leftarrow k - 1$
    return $r$
```

This program computes exactly the same thing as the one in the previous example but is implemented in a different way.

**Proof**: We will show the correctness of the program using loop invariant analysis.

**Loop Invariant**: At the start of each iteration $k$ of the for loop, we always have $n! = k! \cdot r$.

**Initialization**: Prior to the first iteration, by code inspection, at the start of the first iteration, $r = 1$ and $k = n$ as a result of the assignments in Lines 2 and 3.
Therefore, $k! \cdot r = n! \cdot 1 = n!$.
Hence, the loop invariant agrees with what the code does. The loop invariant then holds at the start of the first iteration.

**Maintenance**: Assuming the invariant holds at any iteration $k$, we must show that the loop invariant remains true at the start of the next iteration $k^{'}$. In other words, we must show that the following proposition holds:

**At the start of the iteration $k^{'}$ of the for loop, $n! = k^{'}! \cdot r^{'}$.**

Let us consider an arbitrary iteration $k$. At the start of the iteration $k$, we have $n! = k! \cdot r$ by the loop invariant.
By code inspection, $r$ is updated to $k \cdot r$. Let us denote the new value of $r$ as $r^{'}$. After decrementing $k$ by one, we have $k^{'} = k - 1$ and $k^{'}! \cdot r^{'} = (k - 1)! \cdot (k \cdot r) = k! \cdot r = n!$. Therefore, the loop invariant still holds at the start of the next iteration $k^{'}$.

**Termination**: When the loop exits, we have $k = 0$. Substituting $k = 0$ into the loop invariant, we have $n! = 0! \cdot r$. Since $0! = 1$, it immediately follows that $r = n!$. Therefore, the program correctly returns $n!$ as expected. $\square$

**Example 4.** *Show that the following algorithm searches an array $A[1...n]$ for a key $k$ and correctly returns the index of the key if the search is successful. Otherwise, the algorithm return 0, as an indication that the search is unsuccessful as the key does not exist anywhere in $A[1...n]$.*

**Proof**: We will show the correctness of the program using loop invariant analysis.

**Loop Invariant**: At the start of each iteration $i$ of the for loop, the key $k$ does not exist anywhere in the subarray $A[1...i - 1]$.

**Algorithm 4** implements linear search

```
procedure LINEAR-SEARCH($A[1...n], k$)
    for $i = 1 \to n$ do
        if $A[i] == k$ then
            return $i$
    return 0
```

**Initialization**: At start of the first iteration, $i = 1$. By the loop invariant, at the start of the first iteration, the key $k$ does not exist anywhere in $A[1...i-1] \equiv A[1...1-1] \equiv A[1...0]$, which is an empty array. It is always trivially true that no key ever exists in an empty array. Therefore, the loop invariant holds at the start of the first iteration.

**Maintenance**: Assuming the invariant holds at any iteration $i$, we must show that the loop invariant remains true at the start of the next iteration $i^{'}$. In other words, we must show that the following proposition holds:

**At the start of the iteration $i^{'}$ of the for loop, the key $k$ does not exist anywhere in the subarray $A[1...i^{'}-1]$.**

Let us consider an arbitrary iteration $i$. At the start of the iteration $i$, it follows that the key $k$ does not exist anywhere in the subarray $A[1...i-1]$ by the loop invariant.

If $A[i] \neq k$, we have $i$ incremented by one. In other words, we have $i^{'} = i + 1$. Therefore, at the start of the next iteration $i^{'}$, the key $k$ does not exist anywhere in $A[1...i] \equiv A[1...i^{'}-1]$, reestablishing the loop invariant as required.

If $A[i] = k$, the loop terminates without incrementing $i$. Therefore, we have $i^{'} = i$. Therefore, at this point, the loop invariant that the key does not exist anywhere in $A[1...i-1] \equiv A[1...i^{'}-1]$ still holds as required.

**Termination**: When the loop exits, we have two possible cases to consider.

**Case I**: The loop terminates when $i = n + 1$. Substituting $i = n + 1$ into the loop invariant, we have that the key $k$ does not exist anywhere in $A[1...i-1] \equiv A[1...(n+1)-1] \equiv A[1...n]$, which is the entire array. This means the key $k$ does not exist anywhere in the entire array $A[1...n]$. In this case, the algorithm should return 0, which agrees with what the algorithm does in Line 5. $\square$

**Case II**: The loop terminates at iteration $i$ because $A[i] = k$. In this case, the algorithm should return $i$ as it is where the key $k$ is found in the array $A[1...n]$. By code inspection, the algorithm returns $i$ in Line 4 based on the comparison $A[i] == k$ in Line 3. $\square$