Loop Invariants

Definition

A *loop invariant* for a given loop is a Boolean statement that is true before the first iteration of the loop and is not changed to false during any iteration of the loop. (We call that the inductive condition.) Consequently, the loop invariant is true after the loop terminates. (Do not confuse the loop invariant with the loop condition, which is also a Boolean statement.)

We want to find loop invariants that are useful, that is, help us to analyze code. For example, the condition "1+1 = 2" satifies the definition of a loop invariant, for any loop, even! But it has no practical value.‘

Example

```
 1 int i = n; // input condition: n >= 0
 2 int j = 0;
 3 // loop invariant holds here
 4 while(i > 0)
 5  {
 6   // if the loop invariant holds here,
 7   i = i-1;
 8   j = j+1;
 9   // then the loop invariant holds here.
10  }
11   // thus the loop invariant holds here.
12   cout << j;
```

The goal of this code is to output n. A useful loop invariant is the statement "$i \geq 0$ and $i + j = n$" The invariant is clearly true at line 3. We need to prove that, if the invariant holds at line 6, it holds at line 9. It then must hold at line 11.

At line 3, the loop invariant holds, because $i \geq 0$ by the input condition, and $i + j = i = n$. We need to prove the inductive condition, that is, that if the invariant holds at line 6 during an iteration it holds at line 9 during the same iteration. Suppose the invariant holds at line 6 during one iteration. Hence $i \geq 0$ and $i + j = n$. Let i', j' be the values of those variables after the iteration, that is at line 9. We have $i' = i - 1$ and $j' = j + 1$. By the loop condition, $i > 0$, hence $i \geq 1$ since i is an integer, hence $i' = i - 1 \geq 1 - 1 = 0$. Since $j' = j + 1$ we have $i' + j' = (i - 1) + (j + 1) = i + j = n$. Thus the loop invariant holds at line 9.

We use the loop invariant and the loop condition to prove the code correct. At line 11, since the loop condition is false, we have $i \leq 0$. By the loop invariant, $i \geq 0$, hence $i = 0$. By the loop invariant, $i + j = n$. Thus $j = n - i = n$ at line 11, hence n is the output at line 12.

Multiplication Example

Before our Arabic numeration was invented, the standard algorithm for multiplication we learned in elementary school could not have been used. Instead, people used a method which achieved the correct product by the use of only doubling, halving, and adding, which do not require memorizing multiplication tables. The following C++ function implements this ancient algorithm.

```
int product(int a, int n)
 // input condition: n >= 0
 {
  int s = 0;
  int b = a;
  int m = n;
  // Loop Invariant: m >= 0 and mb + s = na
  while(m > 0)
   {
    if(m%2) // that means m is odd
      s = s+b;
    b = b+b;
    m = m/2; // truncated halving
   }
  return s;
 }
```

We now prove that $na = mb + s$ is a loop invariant. It holds before the first iteration of the loop, since $mn \geq 0$ and $mb + s = na + 0 = na$. We now prove the inductive condition, Assume $mb + s = na$ at the beginning of an iteration. Let m', b', s' be the values of the variables at the end of that iteration. We need to prove that $m' \geq 0$ and $m'b' + s' = na$. Clearly, $m' \geq 0$ because $m > 0$.
Case I: m is even. Then $m' = m/2$, $b' = 2b$, and $s' = s$. We have:

$$\begin{aligned} m'b' + s' &= (m/2)(2b) + s \\ &= mb + s = na \end{aligned}$$

Case II: m is odd. Then $m' = (m-1)/2$, $b' = 2b$, and $s' = s + b$. We have:

$$\begin{aligned} m'b' + s' &= ((m-1)/2)(2b) + s + b \\ &= (m-1)b + s + b \\ &= mb + s \\ &= na \end{aligned}$$

We now prove correctness. Since $m = 0$ and $na = mb + s$ after the last iteration, The function returns $s = na$.