A Proof Example

Consider the following routine that moves the smallest element in the array A to the lower bound location.

```
1     put_smallest_first(A)
2       small := A.lower
3       j := A.lower + 1
4       while j <= A.upper do
5         if A[j] < A[small] then small := j fi
6         j := j + 1
7       end
8       temp := A[small]
9       A[small] := A[A.lower]
10      A[A.lower] := temp
11    end
```

Precondition: $A \neq$ void
Postcondition: $\forall$ k : A.lower+1 .. A.upper • A[A.lower] ≤ A[k]
Loop invariant: $\forall$ k : A.lower .. j – 1 • A[small] ≤ A[k]

Prove the routine is correct.

The parts in teal (this colour) are tutorial comments.

The technical concepts in the slides on Top Down Design are fundamental to writing good programs. To be able to write programs you must answer those questions, as a minimum with English comments. You must answer those questions or else you could not program at all but you can. What you now need to do is to recognize that, and be more formal and careful in documenting your programs.

The loop is the difficult part of programming as shown by the more complex set of questions that need to be answered. Note that Eiffel recognizes this and has the invariant and variant clauses for loops.

From slides 11-12 .. 11-15 you should be able to organize your answer to this question by answering each of the questions, at least in English as a minimum level answer, in the order presented. Higher grades come with better technical competence and better explanations. You should also include diagrams showing initial, middle and final states.

The loop invariant – The difficult part of programming – is given in the question.

Base Case: Is the loop invariant true after loop initialization?

The loop set up is in statements 1 and 2 where the following is true.

small = 1 ^ j = A.lower + 1 (1)

Substitute the relationships in (1) into the LI (loop invariant)

$\forall$ k : A.lower .. A.lower + 1 – 1 • A[1] ≤ A[k]

→ $\forall$ k : A.lower .. A.lower • A[A.lower] ≤ A[k]

The range is one value with k = A.lower and it is certainly true that A[A.lower] ≤ A[A.lower]

Inductive step: Is the loop invariant true after executing the loop once?

Assume LI is true at statement 4 for the general case. We have two cases to consider

Case 1: A[j] < A[small]

Execution of the loop body results in the following relationships.

small′ = j ^ j′ = j + 1 (2)

The loop invariant is the following.

$\forall$ k : A.lower .. j – 1 • A[small] ≤ A[k]

The case condition is A[j] < A[small], as a consequence we have

$\forall$ k : A.lower .. j – 1 • A[small] ≤ A[k] ∧ A[j] < A[small]

Which implies the following – the range can be extended by one to j and A[small] can be replaced by A[j].

$\forall$ k : A.lower .. j • A[j] ≤ A[k]

Now substitute for j from (2) – first j with j′ and the second j wish small′.

$\forall$ k : A.lower .. j′ – 1 • A[small′] ≤ A[k]

This is the form of the loop invariant at the end of the loop, with the new values. As a consequence, executing the loop body in this case preserves the loop invariant.

Case 2: A[j] ≥ A[small]

Execution of the body results in the following relationship.

small′ = small ^ j′ = j + 1 (3)

The loop invariant is the following.

$\forall$ k : A.lower .. j – 1 • A[small] ≤ A[k]

The case condition is A[j] ≥ A[small], as a consequence we have

$\forall$ k : A.lower .. j – 1 • A[small] ≤ A[k] ∧ A[j] ≥ A[small]

Which implies the following – the range can be extended by one to j.

$\forall$ k : A.lower .. j • A[small] ≤ A[k]

Now substitute for j and small from (3) to get the following expression.

$\forall$ k : A.lower .. j′ – 1 • A[small′] ≤ A[k]

This is the form of the loop invariant at the end of the loop, with the new values. As a consequence, executing the loop body in this case preserves the loop invariant.

Since both cases preserve the loop invariant, we conclude that the loop invariant is preserved by execution of the loop body.

Loop termination

The loop terminates because j starts at A.lower+1 and increases by one in both cases of executing the loop body. Eventually, by the rules of arithmetic, j must get bigger than A.upper. As a consequence, the loop terminates.

When the loop terminates the condition is j = A.upper + 1 substitute into the loop invariant to get the following.

$\forall$ k : A.lower .. (A.upper + 1) – 1 • A[small] ≤ A[k]

Simplifying gives the following.

$\forall$ k : A.lower .. A.upper + 1• A[small] ≤ A[k] (4)

Establish the postcondition

At statement 9 the expression (4) is true. The statements 9, 10 and 11 swap the value at A.lower with the value at small.

A′[A.lower] = A[small] ^ A′[small] = A[A.lower] (5)

Substitute (5) into (4)

$\forall$ k : A.lower .. A.upper • A′[A.lower] ≤ A′[k]

But that is the postcondition, as the A' are the final values of A.

QED