**Verification of Finding Max**

Alexander Borgida

2011

**Abbreviation:** for array A, and integers j,k, the notation A[j..k] stands for the set \{A[j],...,A[k]\}, which is empty if k < j.

We are trying to prove that the following program finds the largest element in A[1..n], assuming that this set is not empty.

```c
/* PARAMS: int n ; char A[] */
/* PRE n >=1 */
k:=1
m:=A[1];
while k < n do
  if A[k+1] > m
    then
      m :=A[k+1];
    else
      doNothing ;
    k:=k+1;
end
/* POST: m = max A[1..n] */
```

With loops that end with incrementing a counter (e.g., k in the above case), one good choice of an invariant INV(k) is to replace the final ”n” by ”k” (or something close to it: ”k-1”, ”k+1” – this is where tracing what happens in a few runs of the loop helps). In this case, try

\[
INV : \quad ((k \leq n) \land (m = \text{max}A[1..k]))
\]

So now you have to check:

1. Is INV made true before the loop begins?

2. Is INV maintained by the body of the loop: \(INV \land (k < n) \{\text{body}\} INV\) (because then \(INV \{\text{while } (k<n) \text{ do body end}\} (INV \land \neg (k<n))\) is true by the while-loop inference rule).

3. Does \(INV \land \neg (k < n) \rightarrow POST\)?

We answer these questions in the following order of importance:

3. \(INV \land \neg (k < n) \equiv (k \leq n) \land m = \text{max}A[1..k] \land (k \geq n) \rightarrow m = \text{max}A[1..n] \equiv POST\), so this is fine.

1. Show INV is made true before the loop begins. In Hoare notation, this is just showing:

```
( n > = 1 )
{
  k:=1
  m:=A[1];
}
INV: ( (k <= n) & (m = max A[1..k]) )
```
which can be worked backwards using the assignment rule and logic/algebra as indicated by the following inserted comments:

```c
/* PRE: n >= 1
/*P2: ( (1 <= n) & (A[1] = max A[1..1]) )
   k:=1
/*P1: ( (k <= n) & (A[1] = max A[1..k]) )
   m:=A[1];
/* INV: ( (k <= n) & (m = max A[1..k]) ) */
```

2. Show INV is maintained by the body of the loop, once you get inside (and the condition is true. This is shown by the following comments inserted between the code statements:

```c
/* INV & k < n == k<n & m =max A[1..k] */
if (A[k+1] > m)
   then
      /* now A[k+1] > m & k < n & m = max A[1..k] , by then-rule */
      /* this implies P7 by algebra (k<n implies k+1<=n for integers) and (A[k+1]>m & m>=A[1..k] implies A[k+1]>=max A[1..k+1] */
      /* P7: k+1 <= n & A[k+1]= max A[1..k+1], by assignment rule */
      m :=A[k+1];
      /* P6: same as P3 */
else
   /* now A[k+1] <= m & (k <= n) & m =max A[1..k], by else-rule */
   /* This implies P5 by algebra */
   /* P5 same as P4, by rule for doNothing */
   doNothing
   /* P4 same as P3 */
/*P3: k+1 <= n & m = max A[1..k+1] */
   k := k+1
/* INV: ( (k <= n) & (m = max A[1..k]) ) */
end
```

These three steps can be combined in a single commented version of the program as follows

```c
/* PRE: n >= 1
/*P2: ( (1 <= n) & (A[1] = max A[1..1]) )
   k:=1
/*P1: ( (k <= n) & (A[1] = max A[1..k]) )
   m:=A[1];
/* INV: ( (k <= n) & (m = max A[1..k]) ) */
while (k < n) do
   /* now k < n & (k <= n) & m =max A[1..k] ==
      k<n & m = max A[1..k] */
   if (A[k+1] > m)
      then
         /* now A[k+1] > m & k < n & m = max A[1..k] , by then-rule */
```
/* this implies P7 by algebra (k<n implies k+1<=n for integers) and (A[k+1]>m & m>=A[1..k] implies A[k+1]>=max A[1..k+1] */

/* P7: k+1 <= n & A[k+1]= max A[1..k+1], by assignment rule */
m := A[k+1];
/* P6: same as P3 */
else
  /* now A[k+1] <= m & (k <= n) & m =max A[1..k], by else-rule */
  /* This implies P5 by algebra */

  /* P5 same as P4, by rule for doNothing */
doNothing
  /* P4 same as P3 */
  /*P3: k+1 <= n & m = max A[1..k+1] */
k := k+1
/* INV: ( (k <= n) & (m = max A[1..k]) ) */
end
/* INV & ~(k<n) == ((k <= n) & m=max A[1..k]) & (k >= n) */
/* This implies POST below by simple algebra since k must be n. */
/*POST: m = max A[1..n] */