CSDS 600: Programming Language Concepts, Semantics Written Exercise

Problem 1:
a) State whether the given loop invariants are correct, and if not to give corrected loop invariants
b) For each numbered statement, give the correct *weakest precondition* for the statement
c) Determine whether the weakest precondition for statement 1 is logically inferred from the stated precondition of the code.

Precondition: `n` $\geq$ `0` and `A` contains `n` elements indexed from 0

```
1.   bound = n;
     /* Loop invariant: A[bound,...,(n-1)] is sorted (non-decreasing) and A[bound] >= A[0,...,bound-1] =
2.   while (bound > 0) {
3.     t = 0;
4.     i = 0;
       /* Loop invariant: A[t] is the larges element of [0,...,t] */
5.     while (i < bound-1) {
6.        if (A[i] > A[i+1]) {
7.          swap = A[i];
8.          A[i] = A[i+1];
9.          A[i+1] = swap;
10.         t = i+1;
          }
11.       i++;
       }
12.      bound = t;
    }
```

Postcondition: `A[0]` $\leq$ `A[1]` $\leq$ ...$\leq$ `A[n-1]`

Answer a) The outer loop weakest precondition is correct. The inner loop weakest precondition should be strengthened. Here is my proposed loop invariant: *A[t] $\geq$ A[0..t-1] and A[t..i] is sorted and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

There are multiple ways you can write the inner loop invariant. A student in the class came up with a different loop invariant for the inner loop which is equally correct: *A[bound...n-1] is sorted and A[t..i] is sorted and A[t] $\geq$ A[0..t-1] and A[bound] $\geq$ A[i].*

b) The weakest preconditions for each statement is:

12. *A[t..n-1] is sorted and A[t] $\geq$ A[0..t-1].*

11. *A[t] $\geq$ A[0..t-1] and A[t..i+1] is sorted and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

10. *A[i+1] $\geq$ A[0..i] and A[i+1..i+1] is sorted and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

9. *swap $\geq$ A[0..i] and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

8. *swap $\geq$ A[0..i-1,i+1] and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

7. *A[i] $\geq$ A[0..i+1] and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

6. *A[t] $\geq$ A[0..t-1] and A[t..i] is sorted and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1] and (A[i] < A[i+1] $\rightarrow$ A[i] $\geq$ A[i+1]) and (A[i] $\leq$ A[i+1] $\rightarrow$ A[i] $\leq$ A[i+1]).*

Removing the tautologies gives: *A[t] $\geq$ A[0..t-1] and A[t..i] is sorted and A[bound..n-1] is sorted and A[bound]*

$\geq$ *A[0,...,bound-1].*

5. The same as the inner loop invariant: *A[t] $\geq$ A[0..t-1] and A[t..i] is sorted and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

4. *A[t] $\geq$ A[0..t-1] and A[t..0] is sorted and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].*

3. *A[0] $\geq$ A[0..-1] and A[0..0] is sorted and A[bound..n-1] is sorted and A[bound] $\geq$ A[0,...,bound-1].* This is the same as the outer loop invariant.

2. Same as the outer loop invariant: *A[bound..n-1] is sorted and A[bound] $\geq$ A[0..bound-1].*

1. *A[n..n-1] is sorted and A[n] $\geq$ A[0..n].* Vacuously true because A[n] does not exist assuming A starts indexing at 0.

c) The weakest precondition of 1 is logically inferred from the precondition of the code.

Problem 2: Let M_{state} be a denotational semantic mapping that takes a syntax rule and a state and produces a state. Define the M_{state} mapping for the following two syntax rules *assuming we allow side effects*, that is, assuming expressions and conditions can change the values of variables.

⟨assign⟩ → ⟨var⟩ = ⟨expression⟩
<if> → if <condition> then <statement$_1$> else <statement$_2$>
⟨while⟩ → while ⟨condition⟩ ⟨loop body⟩

Assume we have the following mappings defined:
M_{value} takes a syntax rule and a state and produces a numeric value (or an error condition).
$M_{boolean}$ takes a syntax rule and a state and produces a `true / false` value (or an error condition).
M_{name} takes a syntax rule and produces a name (or an error condition).
Add takes a name, a value, and a state and produces a state that adds the pair **(name, value)** to the state.
Remove takes a name and a state and produces a state that removes any pair that contains the name as the first element.

Also assume you have M_{state}, M_{value}, $M_{boolean}$, and M_{name} defined for ⟨var⟩, ⟨expression⟩, ⟨condition⟩, and ⟨loop body⟩.

Answer:
For the assignment statement, we need to create a new state with a new binding, but the state we use for the binding is the result of any side effect of the right hand side of the assignment statement. Here is the semantic rule from class for the case with no side effect, let s the the initial state:

$$M_{state}(\texttt{<var> = <expression>}, s) = Add(M_{name}(\texttt{<var>}), M_{value}(\texttt{<expression>}, s), Remove(M_{name}(\texttt{<var>}), s))$$

With side-effects, we have to change the state used in *Add* to be the result of any side effects:

$$\begin{aligned} M_{state}(\texttt{<var> = <expression>}, s) = Add(&M_{name}(\texttt{<var>}),\\ &M_{value}(\texttt{<expression>}, s),\\ &Remove(M_{name}(\texttt{<var>}), M_{state}(\texttt{<expression>}, s)) \end{aligned}$$

For the if, before side effects:

M_{state}(if <condition> then <statement$_1$> else <statement$_2$>, s) = {
    if $M_{boolean}$(<condition>, s) = true then
        M_{state}(<statement$_1$>, s)
    else
        M_{state}(<statement$_2$>, s)
}

With side effects, we now have:

M_{state}(if <condition> then <statement$_1$> else <statement$_2$>, s) = {
    if $M_{boolean}$(<condition>, s) = true then
        M_{state}(<statement$_1$>, M_{state}(<condition>, s))
    else
        M_{state}(<statement$_2$>, M_{state}(<condition>, s))
}

For the while, before side effects, the semantic rule from class was the following. The key thing is that the state used to evaluate the loop in the next iteration comes from evaluating the body of the loop in the current state.

M_{state}(`while <condition> <loop body>`, s) = {
    if $M_{boolean}$(`<condition>`, s) = false then
        s
    else
        M_{state}(`while <condition> <loop body>`, M_{state}(`<loop body>`, s))
}

With side effects, the condition can change the state, so there are two places we have the use the new state after it is modified by the condition:

M_{state}(`while <condition> <loop body>`, s) = {
    if $M_{boolean}$(`<condition>`, s) = false then
        M_{state}(`<condition>`, s)
    else
        M_{state}(`while <condition> <loop body>`, M_{state}(`<loop body>`, M_{state}(`<condition>`, s)))
}