Formal Methods in Software Development
Exercise 4 (November 25)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

The result is to be submitted by the deadline stated above *via the Moodle interface* of the course as a `.zip` or `.tgz` file which contains

1. a PDF file with
   - a cover page with the course title, your name, Matrikelnummer, and email address,
   - a section for each part of the exercise with the requested deliverables and optionally any explanations or comments you would like to make;
2. the RISCAL specification (`.txt`) file(s) used in the exercise.

Email submissions are *not* accepted.

Exercise 4: Deriving and Checking Verification Conditions

As in Exercise 1, we consider the following problem: given an array a of $N > 0$ non-negative integer elements, find the maximum element m of a. We claim that this specification is implemented by the following program (fragment):

```
m := 0; i := 0;
while i < N do
{
  if a[i] > m then m := a[i];
  i := i+1;
}
```

The goal of this exercise is to verify this claim by deriving and checking the verification conditions whose validity implies the total correctness of the program with respect to its specification.

1. Take the RISCAL specification file `maximum.txt` which embeds above code in a procedure `maximumElement` and equip this procedure with suitable pre-conditions (`requires`) and post-conditions (`ensures`) that formalize above specification (see Exercise 1). Check (for moderately large values $N > 0$ and $M \geq 0$) that the procedure satisfies the specification.

   Hint: in order to avoid any later confusion between the program variable i and mathematical variables, it is recommended to name quantified variables different from i.

2. Select the operation button "Show/Hide Tasks" to display all tasks related to the specification of the procedure ("Execute specification", "Validate specification" and "Verify specification preconditions"). Validate the specification by executing these tasks (run "Execute specification" with execution option "Silent" switched off to investigate the input/output pairs allowed by your specification; the checks of the other tasks which denote theorems may be performed with option "Silent" switched on).

3. Annotate the loop with suitable invariants (`invariant`) and termination term (`increases`) (since a is a constant, it is not necessary to specify that its value remains unchanged). Rerun the procedure check to ensure that your annotations are not too strong (but they may be too weak to carry the verification).

   Hint: the invariant consists of the following parts:

   - Knowledge about the parameter a that is derived from the precondition (which still holds since a remains unchanged).
   - Basic knowledge about the range of i ($\ldots \leq i \leq \ldots$) and the relationship of i and m ($i = 0 \Rightarrow m = \ldots$).
   - Knowledge about m which arises from the postcondition (which talks about the situation after the termination of the loop when the whole array has been processed) and is adequately generalized (to include the situation before/after every loop iteration when only part of the array has been processed): ($i > 0 \Rightarrow \exists k : 0 \leq k < i \wedge \ldots$) and ($\forall k : 0 \leq k < i \Rightarrow \ldots$).

4. Select the operation button “Show/Hide Tasks” to display all tasks related to the implementation (“Verify correctness of result”, “Verify iteration and recursion”, and “Verify implementation preconditions”) and verify the implementation by checking these tasks. If your annotations are adequate (strong enough but not too strong), then all red tasks turn blue (as demonstrated in class for the “summation” example).

5. Demonstrate your knowledge of the Hoare/Dijkstra calculus by manually deriving from the specification and the loop annotations the verification conditions whose validity implies the total correctness (partial correctness and termination) of the program.

   Hint: pure Hoare calculus reasoning yields five conditions: one for showing that the input condition of the loop (which is different from the input condition of the program!) implies the invariant, one for showing that the invariant and the negation of the loop condition implies the output condition, two for showing that the invariant is preserved and the value of the termination term is decreased for each of the two possible execution paths in the loop body, one for showing that the invariant implies that the value of the termination term does not become negative (if you apply weakest precondition reasoning within the loop body, only one condition is derived from the loop body).

   Do not only give the final verification conditions but show in detail their derivation by application of Hoare calculus respectively the predicate transformer calculus (weakest precondition and strongest postcondition reasoning). *Don’t try to “guess” the condition(s)!*

   Check these conditions in the style of the verification of the “linear search” algorithm presented in class with the help of RISCAL. For this purpose, define predicates `Input`, `Output`, and `Invariant` and a function `Termination`, where (as shown in class) `Invariant` and `Termination` should be parametrized over the program variables. Then define five theorems `A`, `B1`, `B2`, `C`, `D` describing the verification conditions and check these.

The deliverables for this exercise consists of the following items:

1. a nicely formatted copy of the RISCAL specification (included as text, not as screenshots);
2. screenshots of (part of) the RISCAL software illustrating the automatically generated tasks after checking (panel “Tasks”, most (all) of these tasks should be blue);
3. an explicit statement of whether all tasks could be successfully checked or not; if some tasks could not be successfully checked, give screenshots of the RISCAL software after the task has been selected (indicating in the editor area those parts of the program related to the task) and your conjecture why this task failed;
4. a detailed manual derivation of the verification conditions;
5. the outputs of the checks of these verification conditions (included as text, not as screenshots) with explicit statements whether the checks succeeded; if a task failed, give a conjecture why the task failed.

If the check of a theorem fails, attempt to visualize its evaluation and give corresponding screenshots and explanations (see Exercise 1).