Formal Methods in Software Development
Exercise 2 (November 6)

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;
3. the RISC ProofNavigator (`.pn`) file(s) used in the exercise;
4. the proof directories generated by the RISC ProofNavigator.

Email submissions are *not* accepted.

Exercise 2: Deriving, Checking, and Proving Verification Conditions

As in Exercise 1, we consider the following problem: given an array a of positive length N that holds only non-negative integers, find the maximum m of a. We claim that this specification is implemented by the following program (fragment):

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

The goal of this exercise is to formally verify this claim by proving the total correctness of the program with respect to the specification.

1. Take the RISCAL specification file `maximum.txt` which embeds above code in a procedure `maximumValue` and equip this procedure with suitable pre-conditions (`requires`) and post-conditions (`ensures`) that formalize above specification (see Exercise 1). Check (for values $N = 0$ and some $N > 0$) 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. Annotate the loop with suitable invariants (`invariant`) and termination term (`increases`) (since the values a and N are constants, it is not neither necessary nor possible to specify that their values remain unchanged). Rerun the checks to validate the invariant and termination term.

   Hint: the invariant consists of the following parts:

   - Knowledge about the parameters a and N that are derived from the precondition (which still holds since both remain 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 ($\forall k : 0 \leq k < i \Rightarrow \ldots$).

3. Use specification and loop annotations to produce by application of the Hoare calculus the verification conditions whose validity implies the total correctness 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)!*

4. Check the conditions for both $N = 0$ and some $N > 0$ 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.

5. Finally, prove for arbitrary $N \geq 0$, the conditions in the style of the verification of the "linear search" algorithm presented in class with the help of the RISC ProofNavigator. For this, write a declaration file with the following structure

```
newcontext "exercise2";

// arrays as presented in class (except ELEM = INT)
ELEM: TYPE = INT;
...
// program variables and mathematical constants
a: ARR; N: INT;
...
```

In this formalization, extend pre-, post-condition, and invariant by the formula `N = length(a)` (which implies `N >= 0`); again you may omit clauses that state that `a` and `N` remain unchanged.

All proofs can be performed with the commands `expand`, `scatter`, `instantiate`, `split`, and `auto` (it may be sometimes wise to use the `goal` command to switch the goal formula).

The deliverables for this exercise consists of the following items:

1. a detailed derivation of the verification conditions;
2. a nicely formatted copy of the RISCAL specification (included as text, not as screenshots);
3. the outputs of the checks (included as text, not as screenshots) with explicit statements whether the checks succeeded;
4. a (nicely formatted) copy of the ProofNavigator file (included as text, not as screenshots);
5. for each proof of a formula F, a readable screenshot of the RISC ProofNavigator after executing the command `proof` F with explicit statements whether the proof succeeded;
6. if any check gives an error respectively any proof fails, a detailed explanation of the estimated reason of the failure.