Formal Methods in Software Development
Exercise 2 (November 7)

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,
   - the deliverables requested in the description of the exercise,
   - a (nicely formatted) copy of the ProofNavigator file used in the exercise,
   - for each proof of a formula F, a readable screenshot of the RISC ProofNavigator after executing the command `proof` F,
   - an explicit statement whether the proof succeeded,
   - optionally any explanations or comments you would like to make;
2. the RISC ProofNavigator (.pn) file(s) used in the exercise;
3. the proof directories generated by the RISC ProofNavigator.

Email submissions are *not* accepted.

Exercise 2: Searching for the Maximum of an Array

Take the specification

$$\{a = olda \wedge n = oldn \wedge 1 \leq n \leq length(a) \wedge (\forall j : 0 \leq j < n \Rightarrow a[j] \geq 0)\}$$
$$\ldots$$
$$\{a = olda \wedge n = oldn \wedge$$
$$(\forall j : 0 \leq j < n \Rightarrow m \geq a[j]) \wedge (\exists j : 0 \leq j < n \wedge m = a[j])\}$$

which demands to find in an integer array a the maximum m of all elements from the initial segment $a[0 \ldots n-1]$ of length $n \geq 1$ which holds only non-negative integers.

We claim that the specification is implemented by the following program:

```
m := 0; i := 0
while i < n do
  if a[i] > m then
    m := a[i]
  endif
  i := i+1
end
```

The correctness of this claim can be verified by using a loop invariant of the following shape:

$$a = olda \wedge n = oldn \wedge 1 \leq n \leq length(a) \wedge (\forall j : 0 \leq j < n \Rightarrow a[j] \geq 0) \wedge$$
$$(\ldots \leq i \leq \ldots) \wedge (i = 0 \Rightarrow m = \ldots) \wedge$$
$$(\forall j : 0 \leq j < i \Rightarrow \ldots) \wedge (i > 0 \Rightarrow \exists j : 0 \leq j < i \wedge \ldots)$$

Complete the invariant, define a termination term, and use both to produce the five verification conditions[1] for proving the total correctness of the program (one for showing that the input condition of the loop 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). Explicitly show the derivation of the conditions (not only the final result). *Don't try to "guess" the condition(s) but derive them by application of the Hoare axioms respectively of the predicate transformer calculus!*

Please note that the input condition of the loop is not the same as the input condition of the specification!

[1] If you apply weakest precondition reasoning within the loop body, it is only four conditions.

Verify these 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; olda: ARR;
n: INT; oldn: INT;
...
```

Define predicates `Input`, `Output`, and `Invariant` and a value `Term`, where (as shown in class) `Invariant` and `Term` should be parametrized over the program variables. Then define five formulas `A`, `B1`, `B2`, `C`, `D` describing the verification conditions and prove these.

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).