Formal Methods in Software Development
Exercise 1 (April 19)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

March 20, 2007

The result is to be submitted to me by **April 19** (hard deadline) as follows:

- the manual proof can be handed out to me in class as a report (pages stapled) or sent to me per email as a single PDF file, in both cases with a cover sheet that contains your name and "Matrikelnummer";
- the semi-automatic proof is to be sent to me per email that includes as attachments the declarations file and the proof directory (as a .zip or .tgz file) generated by the RISC ProofNavigator.

1 Search for Maximum (Verification, 10 Points)

Take the Hoare triple

$$
\begin{array}{l}
\{a = \mathit{olda} \wedge n = \mathit{oldn} \wedge (\forall j : 0 \leq j < n \Rightarrow a[j] \geq 0) \wedge i = 0 \wedge x = -1\} \\
\textbf{while } i < n \textbf{ do} \\
\quad \textbf{if } a[i] > x \textbf{ then} \\
\quad\quad x := a[i] \\
\quad \textbf{end} \\
\quad i := i + 1 \\
\textbf{end} \\
\{a = \mathit{olda} \wedge n = \mathit{oldn} \wedge \\
\quad ((n = 0 \wedge x = -1) \vee \\
\quad\quad ((\forall j : 0 \leq j < n \Rightarrow a[j] \leq x) \wedge (\exists j : 0 \leq j < n \wedge a[j] = x)))\}
\end{array}
$$

which describes the core proof obligation for a program that searches for the maximum element x in an array of non-negative integers.

This Hoare triple can be verified with the help of the following loop invariant:

$$
\begin{array}{l}
a = \mathit{olda} \wedge n = \mathit{oldn} \wedge (\forall j : 0 \leq j < n \Rightarrow a[j] \geq 0) \wedge i \leq n \wedge \\
((i = 0 \wedge x = -1) \vee \\
\quad ((\forall j : 0 \leq j < i \Rightarrow a[j] \leq x) \wedge (\exists j : 0 \leq j < i \wedge a[j] = x)))
\end{array}
$$

Use this invariant to produce the four verification conditions for proving the partial correctness of the program (one for showing that the input condition 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 for each of the two possible execution paths in the loop body).

Your task is now to verify these five conditions in the style of the verification of the "linear search" algorithm presented in class

1. by a manual proof, and
2. by a semi-automatic proof with the help of the RISC ProofNavigator.

As for the second item, write a declaration file with the following structure

```
newcontext "maxsearch";

// arrays as presented in class (with ELEM set to INT)
...
ELEM: TYPE = INT;
...

// program variables and mathematical constants
a: ARR; olda: ARR;
n: NAT; oldn: NAT;
i: NAT; x: INT;

...
```

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

All proofs can be done with the command `expand`, `scatter`, `split`, `goal`, `auto` only (I recommend to use in some proof states the `goal` command to switch the goal formula, but this is not strictly necessary).