# Automated Reasoning: Tutorial 5

## Introduction

In the following exercise, you will formally verify the correctness of some simple programs using Isabelle's *Hoare_Logic* library. This library allows you to formalise the specifications of programs of a simple programming language in the form of Hoare triples.

The supported programming language includes the following constructs:

- Local variable declaration: `VARS x y z`
- Sequence: `p ; q`
- Skip (do nothing): `SKIP`
- Variable assignment: `x := 0`
- Conditional: `IF cond THEN p ELSE q FI`
- Loop: `WHILE cond INV {invariant} DO p OD`

A program `X` with precondition `P` and postcondition `Q` can be specified as the Hoare triple and, when mechanized in Isabelle, must begin with a local variable declaration `VARS` including at least one local variable, i.e. a Hoare triple is specified in Isabelle as follows:

```
"VARS a {P} X {Q}"
```

Note that a loop invariant must be explicitly specified for each while loop using the `INV` operator.

The automatic Isabelle tactic `vcg`, can be used to extract verification conditions from the Hoare triples and convert them to Isabelle subgoals. The tactic `vcg_simp` combines the capabilities of `vcg` with simplification.

Use the provided Isabelle theory file to carry out the exercise. Note that you can use any pre-defined Isabelle type or function (from the library) in the *program specifications*.

## Exercise

Verify the correctness of the following programs. Some of the examples require that you introduce the appropriate invariant `Inv`.

a). The minimum of two integers `x` and `y`:

```
lemma Min:  "VARS (z ::  int)
    {True}
    IF x ≤ y THEN z := x ELSE z := y FI
    { z = min x y }"
```

b). Iteratively copy an integer variable `x` to `y`:

```
lemma Copy:  "VARS (a ::  int) y
    {0 ≤ x}
    a := x; y := 0;
    WHILE a ≠ 0
    INV { Inv }
    DO y := y + 1 ; a := a - 1 OD
    {x = y}"
-- "Replace Inv with your invariant."
```

c). Iterative multiplication through addition:

```
lemma Multi1:  "VARS (a ::  int) z
    {0 ≤ y}
    a := 0; z := 0;
    WHILE a ≠ y
    INV { Inv }
    DO
       z := z + x ;
       a := a + 1
    OD
    {z = x * y}"
-- "Replace Inv with your invariant."
```

d). Alternative multiplication algorithm:

```
lemma Multi2:  "VARS (z ::  int) y
    {y = y0 ∧ 0 ≤ y}
    z := 0;
    WHILE y ≠ 0
    INV { Inv }
    DO
       z := z + x;
       y := y - 1
    OD
    {z = x * y0}"
-- "Replace Inv with your invariant."
```

e). A factorial algorithm:

```
lemma DownFact:  "VARS (z ::  nat) (y::nat)
    {True}
    z := x; y := 1;
    WHILE z > 0
    INV { Inv }
    DO
       y := y * z ;
       z := z - 1
    OD
    y = fact x"
```

```
-- "Replace Inv with your invariant."
```

f). Integer division of x by y:

```
lemma Div: "VARS (r :: int) d
    {y ≠ 0}
    r := x; d := 0;
    WHILE y ≤ r
    INV { Inv }
    DO
    r := r - y;
    d := d + 1
    OD
    { Postcondition }"
-- "Replace Inv with your invariant."
-- "Replace Postcondition with an appropriate postcondition that reflects
the expected behaviour of the algorithm."
```