1 Factorial

- Write a routine that computes the factorial of its input argument \( n \).
- Annotate the routine with pre and postcondition.
- Prove that your implementation is correct.

\[
\begin{align*}
1 & \text{fact}(n: \text{INTEGER}): \text{INTEGER} \\
2 & \text{require} \ n \geq 0 \\
3 & \text{local} \ i: \text{INTEGER} \\
4 & \text{do} \\
5 & \quad \text{from} \\
6 & \quad i := 0 \\
7 & \quad \text{Result} := 1 \\
8 & \quad \text{until} \ i = n \\
9 & \quad \text{loop} \\
10 & \quad i := i + 1 \\
11 & \quad \text{Result} := \text{Result} \times i \\
12 & \quad \text{end} \\
13 & \text{ensure Result} = n! \text{ end}
\end{align*}
\]

With standard notation, our goal is to prove that the following Hoare triple
is valid.

\[
\begin{align*}
1 & \{ n \geq 0 \} \\
2 & \text{from} \\
3 & \quad i := 0 \\
4 & \quad \text{Result} := 1 \\
5 & \quad \text{until} \ i = n \\
6 & \quad \text{loop} \\
7 & \quad i := i + 1 \\
8 & \quad \text{Result} := \text{Result} \times i \\
9 & \quad \text{end} \\
10 & \{ \text{Result} = n! \}
\end{align*}
\]

Let \( Inv \) denote the loop invariant. The following is a proof outline of a partial
correctness proof, based on the inference rule for loops.

\[
\begin{align*}
1 & \{ n \geq 0 \} \\
2 & \text{from}
\end{align*}
\]
Once we find a suitable invariant, we can verify each block separately, thanks to the composition and the loop inference rules.

To determine the invariant, consider the values of $i$ and $\text{Result}$ over a few iterations:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\text{Result}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
</tbody>
</table>

It should be clear that $\text{Result} = i!$ is an invariant characterizing the loop.

Finally, prove each block correct with backward substitution (the assignment rule). The first block:

\[
\begin{align*}
1 & \{ n \geq 0 \} \\
2 & \{ 1 = 0! \} \\
3 & i := 0 \\
4 & \{ 1 = i! \} \\
5 & \text{Result} := 1 \\
6 & \{ \text{Result} = i! \}
\end{align*}
\]

is correct because indeed $1 = 0!$.

The second block:

\[
\begin{align*}
1 & \{ \text{Result} = i! \land i \neq n \} \\
2 & \{ \text{Result} \ast (i + 1) = (i + 1)! \} \\
3 & i := i + 1 \\
4 & \{ \text{Result} \ast i = i! \} \\
5 & \text{Result} := \text{Result} \ast i \\
6 & \{ \text{Result} = i! \}
\end{align*}
\]

is correct because $\text{Result} = i!$ implies $\text{Result} \ast (i + 1) = (i! \ast (i+1)) = (i+1)!$ by elementary arithmetic.

The third block is also correct, because $\text{Result} = i! \land i = n$ implies $\text{Result} = n!$ by elementary arithmetic.

To prove termination, consider the variant $n - i$. It decreases at every iteration because $i$ increases but $n$ does not change:

\[
\{n - i = x\} i := i + 1 ; \text{Result} := \text{Result} \ast i \{n - i < x\}
\]
Also, \( i \leq n \) is a loop invariant, which implies that \( n - i \geq 0 \), hence the variant has a lower bound. This concludes the termination proof.

2 Primality testing

The following piece of code sets \( pr \) to True iff \( x \) — assumed to be greater than one — is a prime number. Prove correctness.

```
1 { x > 1 }
2 from i := 2 ; pr := True
3 until i ≥ x
4 loop
5 if x mod i = 0 then
6   pr := False
7 end
8 i := i + 1
9 end
10 { (¬ pr ⇒ ∃ y (1 < y < x ∧ x mod y = 0))
11 ∧ (pr ⇒ ∀ y (1 < y < x ⇒ x mod y ≠ 0)) }
```

The proof follows the usual proof outline, based on the inference rule for loops, with \( Inv \) denoting the loop invariant.

```
1 { x > 1 }
2 from i := 2 ; pr := True
3 { Inv }
4 until i ≥ x
5 loop
6 { Inv ∧ i < x }
7 if x mod i = 0 then
8   pr := False
9 end
10 i := i + 1
11 { Inv }
12 end
13 { Inv ∧ i ≥ x }
14 { (¬ pr ⇒ ∃ y (1 < y < x ∧ x mod y = 0))
15 ∧ (pr ⇒ ∀ y (1 < y < x ⇒ x mod y ≠ 0)) }
```

The invariant must imply, together with \( i ≥ x \), the postcondition, hence it is probably very close to it syntactically. Indeed, since the loop proceeds by increasing \( i \) from 2 up until \( x \), a loop invariant is obtained by replacing \( x \) with \( i \) in the postcondition. Another clause in the loop invariant specifies the obvious bounds for \( i \): \( 1 < i ≤ x \).

\[
Inv \triangleq 1 < i ≤ x \land (\neg pr \Rightarrow \exists y (1 < y < i \land x \text{ mod } y = 0)) \\
\land (pr \Rightarrow \forall y (1 < y < i \Rightarrow x \text{ mod } y ≠ 0))
\]

2.1 Initialization

The first block (initialization) corresponds to the triple: 

\[

\]
The backward substitution of Inv yields:
\[ 1 < 2 \leq x \land (\neg \text{True} \Rightarrow \exists y \ (1 < y < 2 \land x \mod y = 0)) \]
\[ \land (\text{True} \Rightarrow \forall y \ (1 < y < 2 \Rightarrow x \mod y \neq 0)) \]

Then:
- \( 2 \leq x \) is equivalent to the precondition \( x > 1 \).
- The first implication holds trivially because its antecedent is \text{False}.
- The second implication holds trivially because the interval \( 1 < y < 2 \) is empty for all integer values of \( y \).

### 2.2 Loop iteration

The second block requires to prove:

1 \{ Inv \land i < x \}
2 \[ \text{if } x \mod i = 0 \text{ then } pr := \text{False} \text{ end} \]
3 \[ i := i + 1 \]
4 \{ Inv \}

Using the inference rule for \text{if}, split the proof into two branches.

#### 2.2.1 Then branch

1 \{ Inv \land i < x \land x \mod i = 0 \}
2 \[ pr := \text{False} \]
3 \[ i := i + 1 \]
4 \{ Inv \}

Backward substitution yields:

1 \{ 1 < i+1 \leq x \land (\neg \text{False} \Rightarrow \exists y \ (1 < y < i+1 \land x \mod y = 0)) \]
2 \[ \land (\text{False} \Rightarrow \forall y \ (1 < y < i+1 \Rightarrow x \mod y \neq 0)) \}

- The clauses \( 1 < i < x \) imply the clause \( 1 < i+1 \leq x \), as we are dealing with integer variables.
- The first implication requires to establish \( \exists y \ (1 < y < i+1 \land x \mod y = 0) \), which is implied by \( x \mod i = 0 \) in the precondition for \( y = i < i+1 \).
- The second implication is trivial as its antecedent is false.

#### 2.2.2 Else branch

1 \{ Inv \land i < x \land x \mod i \neq 0 \}
2 \[ i := i + 1 \]
3 \{ 1 < i \leq x \land (\neg pr \Rightarrow \exists y \ (1 < y < i \land x \mod y = 0)) \]
4 \[ \land (pr \Rightarrow \forall y \ (1 < y < i \Rightarrow x \mod y \neq 0)) \}

4
Backward substitution yields:

\[
1 \{ 1 < i + 1 \leq x \land (\neg pr \Rightarrow \exists y (1 < y < i + 1 \land x \mod y = 0))
\]

\[
2 \land (pr \Rightarrow \forall y (1 < y < i + 1 \Rightarrow x \mod y \neq 0)) \}
\]

First notice that the clauses \(1 < x\) imply the clause \(1 < i + 1 \leq x\), as we are dealing with integer variables. Then, the proof follows a case discussion:

1. Case \(pr = \text{False}\).
   We have to establish only the first implication, as the second has false antecedent. The precondition, for \(pr = \text{False}\), says in particular that \(\exists y (1 < y < i \land x \mod y = 0)\). The value \(y\) that satisfies the existential quantification also satisfies the weaker quantification \(\exists y (1 < y < i + 1 \land x \mod y = 0)\) over the larger interval \((1, i + 1)\).

2. Case \(pr = \text{True}\).
   We have to establish only the second implication, as the first has false antecedent. In the precondition with \(pr = \text{True}\), we combine the facts \(\forall y (1 < y < i \Rightarrow x \mod y \neq 0)\) and \(x \mod i \neq 0\) to get \(\forall y (1 < y < i + 1 \Rightarrow x \mod y \neq 0)\), the stronger quantification over the larger interval \((1, i + 1)\).

2.3 Conclusion
The loop invariant clause \(i \leq x\) and \(i \geq x\) imply \(i = x\). Substituting \(x\) for \(i\) in the other loop invariant clauses yields the postcondition of the program.

2.4 Termination
The variant \(x - i\) and the invariant clause \(1 < i \leq x\) can be combined to prove termination.

3 Least common multiple
Consider a simple program computing the least common multiple (LCM) of two integers \(x, y\), with the following specification.

\[
1 \{ x \geq 1 \land y \geq 1 \}
2 \text{ from } z := 1
3 \text{ until } z \mod x = 0 \land z \mod y = 0
4 \text{ loop } z := z + 1
5 \text{ end}
6 \{ z \mod x = 0 \land z \mod y = 0 \land
7 \forall w (1 \leq w < z \Rightarrow (w \mod x \neq 0 \lor w \mod y \neq 0)) \}
\]

Prove its correctness.
The partial correctness proof follows the usual outline, for a suitable loop invariant \(Inv\).

\[
1 \{ x \geq 1 \land y \geq 1 \}
2 \text{ from } z := 1
3 \{ Inv \}
4 \text{ until } z \mod x = 0 \land z \mod y = 0
5 \text{ loop}
\]
\begin{verbatim}
6 { Inv ∧ (z mod x ≠ 0 ∨ z mod y ≠ 0) }
7   z := z + 1
8 { Inv }
9 end
10 { Inv ∧ z mod x = 0 ∧ z mod y = 0 }
11 { z mod x = 0 ∧ z mod y = 0 ∧
12   ∀ w (1 ≤ w < z ⇒ (w mod x ≠ 0 ∨ w mod y ≠ 0)) }

The loop invariant should mirror the last conjunct of the postcondition, hence:
Inv ≜ ∀ w (1 ≤ w < z ⇒ (w mod x ≠ 0 ∨ w mod y ≠ 0))

3.1 Initialization
Backward substitution of Inv through the from block yields:
∀ w (1 ≤ w < 1 ⇒ (w mod x ≠ 0 ∨ w mod y ≠ 0))
which holds trivially because the interval [1, 1) is empty.

3.2 Loop iteration
The loop body is very simple, hence just apply backward substitution of Inv through z := z + 1 to get:
I' ≜ ∀ w (1 ≤ w < z + 1 ⇒ (w mod x ≠ 0 ∨ w mod y ≠ 0))
Inv implies I' for values of w less than z; combined with the other conjunct (z mod x ≠ 0 ∨ z mod y ≠ 0), it is equivalent to I'.

3.3 Conclusion
Inv and the exit condition z mod x = 0 ∧ z mod y = 0 is exactly the postcondition.

3.4 Termination
Use the variant x*y – z and the invariant x*y – z ≥ 0 to prove termination. (Recall that x*y mod x = x*y mod y = 0).
\end{verbatim}