Program Analysis for Software Security

Seminar 3

Assignment 3

- Please complete the following tasks until Seminar 4.
- The Assignment 3 must be presented at the Seminar 4 (all group members must be in the class).

Find the invariant and check it in Viper

```
method main() { // 07-…
  var M: Int
  var N: Int
  var res: Int

  assume N > 0 && M >= 0

  var m: Int := M
  res := 0

  while (m >= N)
    invariant ??
  {
    m := m - N
    res := res + 1
  }

  assert M == res * N + m
}
```

Find the invariant and check it in Viper

```
method main() { // 08-…
  var n: Int; var m: Int; var res: Int

  assume n >= 0 && m >= 0

  var x: Int := n
  var y: Int := m
  res := 0

  while (x > 0)
    invariant ??
  {
    if (x % 2 == 1) {
      res := res + y
    }
    x := x / 2  // right shift
    y := y * 2  // left shift
  }

  assert res == n * m
}
```

The following Viper program attempts to compute the integer square root of some natural number n. Find a suitable invariant for the line marked with TODO such that the program verifies.

Hint: Notice that the specification admits programs that do not enforce the computation of the integer square root of n. You should still find a suitable invariant.

```
method int_sqrt() {
    var n: Int
    assume n >= 0

    var res: Int

    res := 0
    while ((res + 1) * (res + 1) < n)
        invariant false // TODO
    {
        res := res + 1
    }

    assert res * res <= n  && n <= (res + 1) * (res + 1)
}
```

The following Viper program attempts to compute the integer square root of some natural number n more efficiently. Find a suitable invariant for the line marked with TODO such that the program verifies.

Hint: Notice that the specification admits programs that do not enforce the computation of the integer square root of n. You should still find a suitable invariant.

```
method int_sqrt_fast() {
    var n: Int
    assume n >= 0

    var res: Int

    res := 0
    var x: Int := 1
    while (x < n)
        invariant false // TODO
    {
        x := x + 2 * res + 3
        res := res + 1
    }

    assert res * res <= n && n <= (res + 1) * (res + 1)
}
```

Implement and verify the method below such that it returns the square of any given non-negative integer n.

Your implementation must not use recursion or any arithmetic other than constants and +1. That is, x := 0 and x := x + 1 are allowed. However, x := y + z, x := x * y, and x := 2 * x are not allowed.

You may still use arbitrary arithmetic in assertions and invariants.

```
method square(n: Int) returns (res: Int)
    requires n >= 0
    ensures res == n * n
{
    // TODO
}
```