Formal Methods in Software Engineering
Laboratory 5-6

1. A correctness formula $(\!|P|\!)\,\mathtt{S}\,(\!|Q|\!)$ can be written in Dafny as

```
assume P;
S
assert Q;
```

Here is an example for an instance of the assignment axiom:

```
method m1()
{
    var x: int := *;
    var a: int;
    // (| x*2 > 4 |) a := x*2; (| a > 4|)
    assume x*2 > 4;
    a := x*2;
    assert a > 4;
}
```

An instance for Composition is

```
method mc()
{
    var x, y: int := *,*;
  /*
  (| x + y > 0 |)x := x-1;(|x+y+1 > 0 |) /\ (| x + y + 1 > 0 |)y := y+1;(|x+y > 0 |)
  ------------------------------------------------------------------------------
            (| x + y > 0 |) x := x-1; y := y+1; (|x+y > 0 |)

  */
    // if
    assume x + y > 0;
    x := x-1;
    assert x + y + 1 > 0;
    // and
    // assume x + y + 1 > 0;  // not needed. Why?
```

```
    y := y + 1;
    assert x + y > 0;
    // then
    assume x + y > 0;
    x := x-1;
    y := y + 1;
    assert x + y > 0;
}
```

Write a similar example of instance of the inference rule for if in Floy-Hoare logic.

2. Dafny applies the inference rules automatically. Here is an example how a loop invariant is checked:

```
method m2(x: int, y: int)
requires x*x < y*y
decreases *
{
    var a := x;
    while (a < y)
    invariant a*a <= y*y
    decreases *;
    {
        a := a+1;
    }

}
```

Usually, Dafny tries to prove the termination of loops (total correctness). Since it is not able to prove termination for this case, we added the annotation `decreases *`, in order to say to Dafny to skip termination proof. Note that the nontermination of the loop implies the nontermination of the method.
Find an prove (using Dafny) an invariant of the loop from the following method:

```
method m3(a: int, b: int) returns (q: int, r: int)
    decreases *
{
    q := 0;
    r := a;
    while (r >= b)
    decreases *
    {
        r := r - b;
```

```
        q := q + 1;
    }

}
```

Can you find an expression decreased by the loop (a variant)? If yes, check it by replacing * with that expression. If necessary, you may add preconditions to avoid infinite computations.

3. Find and prove (using Dafny) a loop invariant for the next method:

```
method m4(n: int) returns (s: int)
    requires n >= 0
{
    var i,k : int;
    s := 0;
    k := 1;
    i := 1;
    while (i <= n)
    {
            s := s + k;
            k := k + 2 * i + 1;
            i := i+1;
    }
}
```

4. Sequences in Dafny can be used to specify the values of arrays. Here is a predicate that specifies that m is the largest element of a non-empty sequency:

```
predicate maxSpec(a: seq<int>, m: int)
requires |a| > 0

{
    exists i :: 0 <= i < |a| && m == a[i] &&
    forall i :: 0 <= i < |a| ==> a[i] <= m
}
```

Note that sequences are part of the specification language.
We use this predicate to specify the contract (problem solved by) by a method:

```
method max(a: array<int>) returns (m:int)
requires a.Length > 0
ensures maxSpec(a[..], m)
{
    ...
}
```

We may use invariant specifications (and assertions, lemmata, ...) to prove that body of the method satisfies the contract:

```
method max(a: array<int>) returns (m:int)
requires a.Length > 0
ensures maxSpec(a[..], m)
{
    m := a[0];
    for i := 1 to a.Length
    invariant forall j :: 0 <= j < i ==> a[j] <= m
    invariant exists k :: 0 <= k < i && a[k] == m
    {
        if m < a[i] {
            m := a[i];
        }
    }
}
```

(a) Write a method that computes the two largest elements from an array.

(b) Write a specification (precondition, postcondition) for that method.

(c) Prove that the method satisfies the specification, using Dafny.

5. Write a specification for `m3` and show, using Dafny, that method satisfies it.

6. Write a specification for `m4` and show, using Dafny, that method satisfies it.