Lab session 7: Programming Fundamentals

Lab session 7 consist of nine exercises. Each exercise is worth 1 grade point (you get one point for free). All exercises are about proofs with Dafny. Note that it is very easy to fake a correct submission in Themis (for example, make **false** a precondition or **true** a postcondition of a method), but of course that is not allowed. As a result, your grade in Themis is an upperbound for your real grade, and the teaching assistants will check manually all accepted submissions for 'cheating'/misleading Themis. If such abuse is detected, then the punishment is subtraction of two points per exercise.

Also note that Themis does not run tests (like in previous labs). It only verifies correctness using the Dafny verifier (and when it is proven correct, there is actually no need to test anything). This means that a 'green' judgment means a pass, while a non-pass is actually a 'compilation error' (hence purple, not red). Clearly, this automatically implies that you cannot score partial points.

Problem 1

From Themis you can download the file `exercise1.dfy`, which contains:

```
method exercise1(x:int, y:int, ghost X:int, ghost Y:int)
  returns (p:int, q: int)
requires X>Y && ((x==X && y==Y-2*x) || (x==Y && y==X-2*Y))
ensures p==X && q==Y
{
  // implement (and annotate) yourself
}
```

Implement the body of the method, and annotate it (if needed) such that the Dafny verifier accepts it. Of course, you are not allowed to change the pre/post-condition. Note that `X` and `Y` play, within the body of the method, the role of specification constants, so you are not allowed to use them in the body of the method (in other words, the trivial assignments `p:=X; q:=Y;` are not a solution of this problem).

Problem 2

From Themis you can download the file `exercise2.dfy`, which contains:

```
method exercise2(a: int, n:int) returns (x: int)
ensures x>=n && (x==2*a - n || x==3*n - 2*a + 1)
{
   // implement (and annotate) yourself
}
```

Implement the body of the method, and annotate it (if needed) such that the Dafny verifier accepts it. Of course, you are not allowed to change the pre/post-condition.

Problem 3: inverse of exercise 2

From Themis you can download the file `exercise3.dfy`, which contains:

```
method exercise3(y: int, a: int, n:int) returns (x: int)
requires y>=n && (y == 2*a - n || y == 3*n - 2*a + 1)
ensures x == a
{
  // implement (and annotate) yourself
}
```

As you can see from the specification, this method is (almost) the inverse of the method from exercise 2. Implement the body of the method, and annotate it (if needed) such that the Dafny verifier accepts it. Of course, you are not allowed to change the pre/post-condition. You are not allowed to use `a` in the body of the method (it plays the role of a specification constant). So, the trivial assignment `x := a` is not a valid solution of this problem.

Problem 4

From Themis you can download the file `exercise4.dfy`, which contains:

```
method preserveSum(x: int, y:int, a:int) returns (p: int, q: int)
ensures p + q == x + y
{
  p := x + a;
  q := y - a;
}

method preserveSumProduct(x: int, y:int) returns (p: int, q: int)
ensures p + q == x + y && p*q < 0
{
   // implement (and annotate) yourself
}
```

You must implement the body of the method `preserveSumProduct`, and annotate it (if needed) such that the Dafny verifier accepts it. You may use the method `preserveSum` as inspiration for your solution.

Problem 5: Recurrence

From Themis you can download the file `recurrence.dfy`, which contains:

```
function f(n: nat): int
{
  if n <= 1 then 1-n else 2*f(n-1) + f(n-2)
}
```

```
method computeF(n: nat) returns (a: int)
ensures  a == f(n)
{
  // implement and annotate yourself
}
```

You must implement the body of the method `computeF`, which needs to contain a while loop (so no recursion). Annotate your solution such that the Dafny verifier accepts it.

Problem 6: mystery

From Themis you can download the file `mystery.dfy`, which contains:

```
method mystery(a: nat) returns (q:nat, r:nat)
ensures ???
{
  q,r := 0, a + 1;
  while r >= 4
  {
    q := q + r/4;
    r := r/4 + r%4;
  }
  r := r - 1;
}
```

Figure out what this method computes, supply the postcondition, and annotate the body such that the Dafny verifier accepts it.

Problem 7: Divisible by 3

A natural number is divisible by 3, if the sum of its digits is a multiple of 3. In this exercise you will prove this property.
From Themis you can download the file `div3.dfy`, which contains:

```
function sumDigits(n: nat) : nat {
  if n < 10 then n else n%10 + sumDigits(n/10)
}

lemma {:induction false} div3Lemma(n: nat)
ensures sumDigits(n)%3 == n%3
{
  // 'implement' this proof
}
```

Make the proof complete. Note that your are not allowed to remove `{:induction false}`, so you are not allowed to enable Dafny's automatic induction strategy.

Problem 8: Sum of cubes

It is well known that $\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$. In this exercise you will prove this identity. From Themis you can download the file `cubesum.dfy`, which contains:

```
function sumCubes(n: nat) : nat {
  if n == 0 then 0 else n*n*n + sumCubes(n-1)
}

lemma {:induction false} sumLemma(n: nat)
ensures sumCubes(n) == n*n*(n+1)*(n+1)/4
{
  // 'implement' this proof
}
```

Make the proof complete. Note that your are not allowed to remove `{:induction false}`, so you are not allowed to enable Dafny's automatic induction strategy.

Problem 9: exponentiation

From Themis you can download the file `exp.dfy`, which contains:

```
function exp(a:nat, e:nat): nat {
    if e==0 then 1 else a*exp(a,e-1)
}

function fastExp(a:nat, e:nat): nat
{
  if e==0 then 1
  else if e%2==0
       then fastExp(a*a, e/2)
       else a*fastExp(a, e-1)
}

lemma equalExp(a: nat, e: nat)
ensures exp(a,e) == fastExp(a,e)
{
  // 'implement' this proof
}
```

Make the proof complete such that the Dafny verifier accepts it.