Formal Methods in Software Engineering
Laboratory 3 (Coinduction)

1. Read Section 5.14.2 (Coinductive datatypes) and Section 5.14.3 Dafny Reference Manual (https://dafny.org/dafny/DafnyRef/DafnyRef).
2. Define in Dafny the conatural numbers as a coinductive datatype and use this for
   (a) explaining why the following coinductive property does not hold;

```
lemma ConstructorConat(n: Conat)
  ensures n != Succ(n)
{
  // why?
}
```

   (b) showing that the constructor successor is injective
   (c) defining the constant (as a corecursive function) $\infty = \mathsf{s}(\mathsf{s}(\ldots))$;
   (d) defining the addition of conaturals;
   (e) showing thad adding ∞ to itself it remains unchanged.
3. Define the parametric streams as coinductive datatype using the rule

$$\frac{s}{\mathsf{cons}(a, s)}\ a \in A$$

   where s ranges over streams, and use it for
   (a) corecursively defining the pointwise addition of two streams of integers $add(s, s')$;
   (b) defining a parametric integer constant stream $cnst(n)$;
   (c) proving by coinduction that $add(s, cnst(0)) = s$;
   (d) defining coinductively the predicate $leq(s, s')$;
   (e) defining the stream $blink$;
   (f) proving by coinduction that $leq(cnst(0), blink)$;
   (g) defining a function that "zip" two streams; and
   (h) and proving that zipping $cnst(0)$ and $cnst(1)$ you ontain $blink$.

A Coinductive sets in Dafny

$$\frac{}{\mathsf{0}} \qquad \frac{n}{\mathsf{s}(n)}$$

```
codatatype Conat = Zero | Succ(Pred: Conat)  //coinductiv
```

B Corecursive function

$add(\mathsf{0}, n) = n$
$add(\mathsf{s}(m), n) = \mathsf{s}(add(m, n))$

```
function add(m: Conat, n: Conat) : Conat
{
match m
case Zero => n
case Succ(m') => Succ(add(m', n))
}
```

C Coinductive predicate

$lt(0, \mathsf{s}(n))$
$lt(\mathsf{s}(m), \mathsf{s}(n)) \iff lt(m, n)$
$\neg lt(m, n)$ otherwise

```
greatest predicate lt(m: Conat, n: Conat) // it fails as a predicate
{
  match m
    case Zero => n.Succ?
    case Succ(m') => n.Succ? && lt(m', n.Pred)
}
```

D Proofs by coinduction

```
greatest lemma AddZero(m: Conat)  // it fails as a lemma
  ensures add(m, Zero) == m
{
  //
}
```