Homework 4

Functional Programming (ITI0212)

due: 2021.05.10

Place your solutions in a module named `Homework4` in a file with path `homework/Homework4.idr` within your `iti0212-2021` repository on the TalTech GitLab server (`https://gitlab.cs.ttu.ee/`). Your solutions will be pulled automatically for marking. At the start of the file include a comment containing your name and the Idris version you are using. Precede each problem's solution with a comment specifying the problem number.

Problem 1

Convince Idris that the sum of any number and itself is even:

```
double_even  :  (n : Nat) -> Even (n + n)
```

Hint: in the successor case it may help to find a proof of the type:

```
    lemma  :  Even $ S (S (n + n))
```

and `transport` it in the indexed type `Even` along the equality:

```
    path  :  S (S (n + n)) = (S n + S n)
```

Problem 2

Convince Idris that the product of an even number and any number is even:

```
even_times_any  :  (m , n : Nat) -> Even m -> Even (m * n)
```

Hint: in the `SS_even` case it may help to find a proof of the type:

```
    lemma  :  Even $ (n + n) + (m * n)
```

and `transport` it in the indexed type `Even` along the equality:

```
    path  :  (n + n) + (m * n) = n + (n + (m * n))
```

Problem 3

Using the interpretations of `And`, `Or` and `Not` from lecture 13, convince Idris of the following *de Morgan laws*:

```
dm1  :  Not a `Or` Not b -> Not (a `And` b)

dm2  :  Not a `And` Not b -> Not (a `Or` b)
```

Problem 4

Using the interpretation of `Some` from lecture 13, convince Idris that every even number is the double of some other number:

```
evens_are_doubles  :  Even n -> Some Nat $ \ m => m + m = n
```

Problem 5
Write a function with an *auto-implicit* constraint that returns half of an even number:

```
half  :  (n : Nat) -> {auto even : Even n} -> Nat
```

For example:

```
> half 0
0
> half 2
1
> half 42
21
> half 3
Error: Can't find an implementation for Even 3.
```

Hint: you can use your function from problem 4 to make this a one-liner.

Problem 6
Show that the negation of a decidable predicate is decidable:

```
dec_not  :  {p : a -> Type} -> Dec (p x) -> Dec (Not $ p x)
```

Then show that the conjunction of decidable predicates is decidable:

```
dec_and  :  {p , q : a -> Type} -> Dec (p x) -> Dec (q x) ->
  Dec (p x `And` q x)
```