Lab 8: Indexed types and dependent functions

Functional Programming (ITI0212)

This week we are learning about indexed type families and dependent functions. An *indexed type family* (or "indexed type" or "dependent type (family)") is a type constructor where the resulting types are indexed by (or "depend on") elements of another type.

We met the *finite types* `Fin n`, whose elements are finite prefixes of the natural numbers bounded by `n`. We also met the *vector types* `Vect` α `n`, whose elements are length `n` sequences of αs. Both of these are indexed by the type `Nat`. We also saw how the *dependent pair types* `Sigma a b` generalize the ordinary pair types `Prod a b`, in that the type of the second factor is indexed by the first factor.

A *dependent function* is one where the type of the result can depend on not only the type, but also the value, of the argument. The type constructor for dependent functions is built into Lean and is what makes its type system so expressive.

Task 1
Write a *type isomorphism* between the types `Bool` and `Fin 2`, so that:

```
(fin_2_bool ∘ bool_2_fin) true
true : Bool
(fin_2_bool ∘ bool_2_fin) false
false : Bool
(bool_2_fin ∘ fin_2_bool) 0
0 : Fin 2
(bool_2_fin ∘ fin_2_bool) 1
1 : Fin 2
```

Remember that $\circ$ is function composition. *Question:* how many isomorphisms are there between these two types?

Task 2
Let's practise writing some functions for the `Vect` type we have defined in the lecture:

```
inductive Vect (α : Type) : (n : Nat) → Type where
  | nil : Vect α 0
  | cons : α → Vect α n → Vect α (n + 1)
```

First, write the *map* function for vectors. If you're stuck, look at the definition of *map* for `List`s.

Now, write the *unzip* function for vectors which splits a `Vect` of pairs into a pair of `Vect`s. *Hint*: the type signature of this function: `Vect` $\alpha \times \beta$ `n` $\to$ `Vect` α `n` $\times$ `Vect` β `n`.

Finally, write the *reverse* function for vectors. *Hint*: you need a helper function which appends an element to the end of a vector, typically called *snoc* (cons read backwards).

Task 3
Write the following function, which returns the element of a finite type having the

same “size” as its argument, and that is the “largest” element of its type: Note that this is a *dependent function* because the result *type* depends on the argument *value*.

For example:

```
as_top 0
0 : Fin 1
as_top 1
1 : Fin 2
as_top 2
2 : Fin 3
```

Task 4
Write a function `ind_pair` that converts a `Prod` into a `Sigma` with the same elements in the same order.

For example:

```
ind_pair ("hello" , 42)
⟨ "hello", 42 ⟩
ind_pair (true , ())
⟨ true , () ⟩
```

Task 5
`List` types and `Vect` types are both *finite sequence types*, made from constructors called `nil` and `cons`, and informally, we can think of a `Vect` as a `List` that knows its length.

Forgetting things is usually pretty easy. Write a function that converts a `Vect` into the `List` containing the same elements in the same order.

Learning things is often a little harder than forgetting them. Write a function that converts a `List` into the `Vect` containing the same elements in the same order.

Hint: `learn_length` will need to be a *dependent function* because the result *type* depends on the argument *value*.