CIS 5000: Software Foundations Quiz 3 September 18, 2025

Name (printed): _____ Penn ID (8-digit #): _____

Please write your name as it is displayed on Gradescope. Indicate your answers with a cross mark ☒. To correct a mistake, fill in the incorrect box (like this: ■) and make a cross in the correct box.

1 Recall the definition of `nat` (repeated on the handout).

(a) What does "the constructor `S` is injective" mean?

- ☐ `n = m -> S n = S m`
- ☒ `S n = S m -> n = m`
- ☐ `forall n, S n <> O`
- ☐ None of these

(b) What does "the constructors are disjoint" mean?

- ☐ `n = m -> S n = S m`
- ☐ `S n = S m -> n = m`
- ☒ `forall n, S n <> O`
- ☐ None of these

2 You have hypothesis `H : O = S n` in your context. Which tactic should you use?

- ☐ `rewrite H`
- ☐ `apply H`
- ☒ `discriminate H`
- ☐ `inversion H`

3 Consider the `repeat` function (see the handout).

(a) What is the type of `repeat`?

- ☐ `nat -> nat -> list nat`
- ☐ `X -> nat -> list X`
- ☐ `forall (X Y : Type), X -> nat -> list Y`
- ☒ `forall (X : Type), X -> nat -> list X`
- ☐ Ill-typed

(b) What is the type of `repeat nat 1 2`?

- ☒ `list nat`
- ☐ `list (list nat)`
- ☐ `forall (X : Type), X -> nat -> list X`
- ☐ Ill-typed

4 Recall the `map` function (see the handout).

(a) What is the type of `@map`?

- ☐ `forall X Y : Type, X -> Y -> list X -> list Y`
- ☐ `X -> Y -> list X -> list Y`
- ☒ `forall X Y : Type, (X -> Y) -> list X -> list Y`
- ☐ `forall X : Type, (X -> X) -> list X -> list X`

(b) Recall that `even` has type `nat -> bool`. What is the type of `map even`?

- ☐ `forall X Y : Type, (X -> Y) -> list X -> list Y`
- ☒ `list nat -> list bool`
- ☐ `list nat -> list Y`
- ☐ Ill-typed

(c) Which of the following are valid ways to call `@map`?

- ☐ `@map nat bool even [1;2;3] true`
- ☒ `@map nat bool even [1;2;3]`
- ☒ `@map nat bool`
- ☐ `@map even [1;2;3]`
- ☐ `@map even [true;true;false]`

For Reference

```
  Inductive nat: Type :=
  | O
  | S (n: nat).

Fixpoint repeat (X : Type) (x : X) (count : nat) : list X :=
  match count with
  | 0 => @nil X
  | S count' => @cons X x (repeat X x count')
  end.

Fixpoint map {X Y : Type} (f : X->Y) (l : list X)
             : list Y :=
  match l with
  | []     => []
  | h :: t => (f h) :: (map f t)
  end.
```