(* Lecture 02 Exercises *)

(* infer some type arguments automatically *)

Set Implicit Arguments.

Inductive list (A: Set) : Set :=
| nil : list A |
| cons : A -> list A -> list A. |

Arguments nil A.

Fixpoint length (A: Set) (l: list A) : nat :=
match l with
| nil => O  |
| cons x xs => S (length xs)  |
end.

(* add one list to the end of another *)

Fixpoint app (A: Set) (l1: list A) (l2: list A) : list A :=
match l1 with
| nil => l2  |
| cons x xs => cons x (app xs l2)  |
end.

Theorem app_nil:
forall A (l: list A),  app l nil = l. Proof.  intros.  induction l.  + simpl. reflexivity.  + simpl. rewrite IHl. reflexivity. Qed.

Theorem app_assoc:
forall A (l1 l2 l3: list A),  app (app l1 l2) l3 = app l1 (app l2 l3). Proof.  intros.  induction l1.  + simpl. reflexivity.  + simpl. rewrite IHl1. reflexivity. Qed.

(* simple but inefficient way to reverse a list *)

Fixpoint rev (A: Set) (l: list A) : list A :=
match l with
| nil => nil  |
| cons x xs => app (rev xs) (cons x nil)  |
end.

Theorem fast_rev_ok_snoc:
forall A (l: list A),  fast_rev l = rev_snoc l. Proof.  (* TODO −− you will need to define a helper lemma very similar to how we proved fast_ref_ok *)
Admitted.

(* useful in proving rev_length below *)

Lemma plus_1_S:
forall n,  plus n 1 = S n. Proof.  intros.  induction n.  + simpl. reflexivity.  + simpl. rewrite IHn. reflexivity. Qed.

Lemma rev_length:
forall A (l: list A),  length (rev l) = length l. Proof.  (* TODO −− you will need to define a helper lemma that relates length and app *)
Admitted.

(* simple but inefficient way to reverse a list *)

Fixpoint rev_snoc (A: Set) (l: list A) : list A :=
match l with
| nil => nil  |
| cons x xs => cons x (app xs (snoc l x))  |
end.

Lemma fast_rev_ok_snoc:
forall A (l: list A),  fast_rev l = rev_snoc l. Proof.  (* TODO −− you will need to define a helper lemma that relates rev and app, its proof should use app_assoc *)
Admitted.

(* useful in proving rev_length below *)

Lemma rev_involutive:
forall A (l: list A),  rev (rev l) = l. Proof.  (* TODO −− you will need to define a helper lemma that relates rev and app *)
Admitted.

Definition fast_rev (A: Set) (l: list A) : list A :=
fast_rev_aux l nil.

Theorem snoc_app_singleton:
forall A (l: list A) (x: A),  snoc l x = app l (cons x nil). Proof.  (* TODO *)
Admitted.