Chapter 1

Library Coq_assistant_preuve_TI

```
Require Import List String.

Import ListNotations.
Open Scope list_scope.
Open Scope string_scope.

Check [4;5;6].
Check ["Anne";"42"].

Section Insertion_sort.

Variable A : Type.
Variable leb : A → A → bool.

Fixpoint insert (a:A) (l: list A) : list A :=
match l with
  | [] ⇒ [a]
  | b::l0 ⇒ if leb a b
    then a::l
    else b::insert a l0
  end.

Fixpoint sort(l:list A) : list A :=
match l with
  | [] ⇒ []
  | a::l0 ⇒ insert a (sort l0)
end.

Require Import Coq.Sorting.Permutation.

Search (Permutation ?x ?y →
  Permutation ?y ?z →
    Permutation ?x ?z).

Inductive Sorted : list A → Prop :=
  | ls_0 : Sorted []
```

```
  | ls_1 : ∀ a, Sorted [a]
  | ls_2 : ∀ a b l, (leb a b = true) →
      Sorted (b::l) → Sorted (a::b::l).

Definition Sort_spec (f:list A → list A) :=
  ∀ l, let l1 := f l in
    Permutation l1 l ∧ Sorted l1.

Definition Total(comp : A → A → bool) : Prop :=
  ∀ a b, comp a b = false → comp b a = true.

Hypothesis leb_total : Total leb.

Lemma insert_perm :
  ∀ x l, Permutation (x::l) (insert x l).
Proof.
induction l.
- reflexivity.
- cbn.
  case (leb x a).
    + reflexivity.
    + transitivity (a::x::l).
      × Search (Permutation (?x::?y::?l) (?y::?x::?l)).
        apply perm_swap.
      × auto.
Qed.

Lemma insert_sorted : ∀ x l,
  Sorted l → Sorted (insert x l ).
Proof.
intros x l l_sorted.
induction l_sorted.
- simpl. apply ls_1.
- simpl. destruct (leb x a) eqn:x_le_a.
    + apply ls_2.
      × assumption.
      × apply ls_1.
    + apply ls_2.
      × apply leb_total in x_le_a.
        assumption.
      × apply ls_1.
- simpl.
  destruct (leb x a) eqn:x_le_a.
  + apply ls_2.
    × assumption.
    × apply ls_2.
```

```
      ** assumption.
      ** assumption.
  + apply leb_total in x_le_a.
    destruct (leb x b) eqn:x_le_b.
    × apply ls_2.
      ** assumption.
      ** apply ls_2.
        *** assumption.
        *** assumption.
    × apply ls_2.
      ** assumption.
      ** simpl in IHl_sorted. destruct (leb x b).
        *** discriminate.
        *** assumption.
Qed.

Lemma sort_perm : ∀ l, Permutation (sort l) l.
Proof.
induction l.
- simpl. reflexivity.
- transitivity(a::sort l); auto;
  symmetry;apply insert_perm.
Qed.

Lemma sort_sorted: ∀ l, Sorted (sort l).
Proof.
  induction l.
  - simpl. apply ls_0.
  - simpl. apply insert_sorted. assumption.
Qed.

Theorem sort_correct: Sort_spec sort.
Proof.
  split.
  - apply sort_perm.
  - apply sort_sorted.
Qed.

End Insertion_sort.

Check sort.

Check sort_correct.

Lemma nat_leb_total : Total nat Nat.leb.
Proof.
  red; induction a; destruct b; simpl;
  try discriminate;auto.
```

```
Qed.

Check sort_correct nat Nat.leb nat_leb_total.

Compute sort nat Nat.leb [8;6;9;3;7;8].

Definition string_leb s1 s2 :=
    Nat.leb (length s1) (length s2).

Check string_leb.

Lemma string_leb_total : Total string string_leb.
Proof.
unfold Total.
intros a b. unfold string_leb.
apply nat_leb_total.
Qed.

Check sort_correct string string_leb string_leb_total.

Compute sort string string_leb ["cb";"abcd";"z";"cb";"";"dab"].
```