Chapter 1

Library main

```
Require Import Coq.Arith.PeanoNat.
Require Import Lia.
Inductive deck : nat → Set :=
  | nild : deck 0
  | consd : ∀ n : nat, bool → deck n → deck (S n).
Fixpoint flip {n : nat} (d : deck n) : deck n :=
  match d with
    | nild ⇒ nild
    | consd i b d' ⇒ consd i (negb b) (flip d')
  end.
Fixpoint count_ups {n : nat} (d : deck n) : nat :=
  match d with
    | nild ⇒ 0
    | consd _ true d' ⇒ S (count_ups d')
    | consd _ false d' ⇒ (count_ups d')
  end.
Lemma flip_preserves_length : ∀ n (d: deck n),
  flip d = flip d :> deck n.
Proof.
  intros n d.
  reflexivity.
Qed.
Lemma flip_involutive : ∀ n (d: deck n),
  flip (flip d) = d.
Proof.
  intros n d.
  induction d as [| n' b d' IHd].
  -
    simpl. reflexivity.
  -
    simpl. rewrite IHd.
    destruct b; reflexivity.
Qed.

Fixpoint split {n : nat} (d : deck n) (m : nat) (H : m ≤ n) :
  deck m × deck (n - m) :=
  match d in deck n' return ∀ m', m' ≤ n' → deck m' × deck (n' - m') with
  | nild ⇒ fun m' H' ⇒
      match m' as m'' return m'' ≤ 0 → deck m'' × deck (0 - m'') with
      | 0 ⇒ fun _ ⇒ (nild, nild)
      | S m'' ⇒ fun H'' ⇒ match Nat.nle_succ_0 m'' H'' with end
      end H'
  | consd n' b d' ⇒ fun m' H' ⇒
      match m' as m'' return m'' ≤ S n' → deck m'' × deck (S n' - m'') with
      | 0 ⇒ fun _ ⇒
          (nild, consd n' b d')
      | S m'' ⇒ fun H'' ⇒
          let H''' : m'' ≤ n' := le_S_n m'' n' H'' in
          let (d1, d2) := split d' m'' H''' in
          (consd m'' b d1 , d2)
      end H'
  end m H.

Lemma count_ups_flip : ∀ n (d : deck n),
  count_ups (flip d) + count_ups d = n.
Proof.
  intros n d. induction d as [| n' b d' IHd]; simpl.
  - reflexivity.
  - destruct b; simpl; lia.
Qed.

Lemma split_preserves_count : ∀ n m (H : m ≤ n) (d : deck n),
  let (d1, d2) := split d m H in
  count_ups d = count_ups d1 + count_ups d2.
Proof.
  intros n m H d.
  generalize dependent m.
  induction d as [| n' b d' IHd].
  -
    intros m H.
    destruct m.
    + simpl. reflexivity.
    + exfalso. apply (Nat.nle_succ_0 m H).
  -
```

```
    intros m H.
    destruct m.
    + simpl. reflexivity.
    + simpl.
      destruct (split d' m (le_S_n m n' H)) as [d1 d2] eqn:Hsplit.
      specialize (IHd m (le_S_n m n' H)).
      rewrite Hsplit in IHd.
      destruct b; simpl; lia.
Qed.

Lemma split_sizes : ∀ n m (H : m ≤ n) (d : deck n),
  let (d1, d2) := split d m H in
  True.
Proof.
  intros n m H d.
  destruct (split d m H) as [d1 d2].
  trivial.
Qed.

Theorem equal_ups :
  ∀ (n : nat) (m : nat) (H: m ≤ n) (d: deck n),
  count_ups d = m →
  let (d1, d2) := split d m H in
  count_ups (flip d1) = count_ups d2.
Proof.
  intros n m H d Hcount.

  assert (Hpreserve: let (d1, d2) := split d m H in count_ups d = count_ups d1 + count_ups
d2).
  { apply split_preserves_count. }


  destruct (split d m H) as [d1 d2] eqn:Hsplit.
  simpl in Hpreserve.

  rewrite Hcount in Hpreserve.

  pose proof (count_ups_flip m d1) as Hflip.

  lia.
Qed.
```