1.

代码：

```
Fixpoint max' (n : nat)(l : list nat) : nat :=
  match l with
  | nil => n
  | h :: t =>if leb h n then max' n t
             else max' h t
  end.

Definition max (L : list nat) : option nat :=
  match L with
  | nil  => None
  | h1 :: t1 => Some (max' h1 t1 )
  end.

Example test_max1: max [1;2;3;4;100] = Some 100.
Proof. simpl. reflexivity. Qed.

Example test_max2: max [1;3;5;7] = Some 7.
Proof. simpl. reflexivity. Qed.

Example test_max3: max [2;4;6;6] = Some 6.
Proof. simpl. reflexivity. Qed.

Example test_max4: max [] = None.
Proof. simpl. reflexivity. Qed.
```

运行结果：

```
Fixpoint max' (n : nat)(l : list nat) : nat :=
  match l with
  | nil => n
  | h :: t =>if  leb h n then max' n t
           else max' h t
  end.

Definition max (L : list nat) : option nat :=
  match L with
  | nil  =>None
  | h1 :: t1 => Some (max' h1 t1 )
  end.

Example test_max1: max [1;2;3;4;100] = Some 100.
Proof. simpl. reflexivity. Qed.

Example test_max2: max [1;3;5;7] = Some 7.
Proof. simpl. reflexivity. Qed.

Example test_max3: max [2;4;6;6] = Some 6.
Proof. simpl. reflexivity. Qed.

Example test_max4: max [] = None.
Proof. simpl. reflexivity. Qed.
```

分析：

并没有想到直接natlist->natoption的函数构造，于是先递归构造一个`max‘`得到整个数列中的最大值，这里多传入了一个类型为`nat`的参数`n`，便于后续函数的封装。之后再定义函数`max`，判断传入的列表是否为空，如果是空，则返回`None`，否则返回`Some (max'(...))`。其中这里第一个参数即为L的第一个元素，因为前面已经排除了空集的可能。

2.

代码：

```
Fixpoint max'' (L : list nat) : nat :=
  match L with
  | [] => 0
```

```
  | h :: t => match t with
                | [] => h
                | _ => if leb h (max'' t) then max'' t
                          else h
                end
  end.

Definition maxPair (L : list nat) : nat * nat :=
   ((max'' (filter odd L)),  (max'' (filter even L))).

Example test_maxPair1: maxPair [1;2;3;4;5;6;7] = (7, 6).
Proof. simpl. reflexivity. Qed.

Example test_maxPair2: maxPair [1;3;5;7] = (7, 0).
Proof. simpl. reflexivity. Qed.

Example test_maxPair3: maxPair [2;4;6] = (0, 6).
Proof. simpl. reflexivity. Qed.

Example test_maxPair4: maxPair [] = (0, 0).
Proof. simpl. reflexivity. Qed.
```

运行结果：

```
Fixpoint max'' (L : list nat) : nat :=
  match L with
  | [] => 0
  | h :: t => match t with
              | [] => h
              | _ => if leb h (max'' t) then max'' t
                     else h
              end
  end.

Definition maxPair (L : list nat) : nat * nat :=
   ((max'' (filter odd L)),  (max'' (filter even L))).

Example test_maxPair1: maxPair [1;2;3;4;5;6;7] = (7, 6).
Proof. simpl. reflexivity. Qed.

Example test_maxPair2: maxPair [1;3;5;7] = (7, 0).
Proof. simpl. reflexivity. Qed.

Example test_maxPair3: maxPair [2;4;6] = (0, 6).
Proof. simpl. reflexivity. Qed.

Example test_maxPair4: maxPair [] = (0, 0).
Proof. simpl. reflexivity. Qed.
```

分析：

这里用到了`Poly.v`中的`filter`函数，以及`odd`和`even`来分别形成只包含奇数和只包含偶数的列表，再利用构造的函数`max''`：如果是空则按要求返回0；否则返回其中最大值。

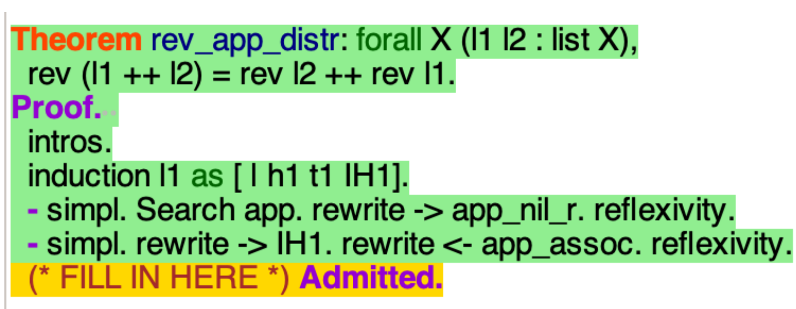
3 & 4.

代码：

```
(** **** Exercise: 2 stars, standard (more_poly_exercises)

    Here are some slightly more interesting ones... *)

Theorem rev_app_distr: forall X (l1 l2 : list X),
```

```
  rev (l1 ++ l2) = rev l2 ++ rev l1.
Proof.
  intros.
  induction l1 as [ | h1 t1 IH1].
  - simpl. Search app. rewrite -> app_nil_r. reflexivity.
  - simpl. rewrite -> IH1. rewrite <- app_assoc. reflexivity.
  (* FILL IN HERE *) Admitted.

Theorem rev_involutive : forall X : Type, forall l : list X,
  rev (rev l) = l.
Proof.
  intros.
  induction l as [ | h t IH].
  - simpl. reflexivity.
  -  simpl.  rewrite -> rev_app_distr. simpl. rewrite -> IH. reflexivity.
  (* FILL IN HERE *) Admitted.
(** [] *)
```

运行结果：



```
Theorem rev_app_distr: forall X (l1 l2 : list X),
  rev (l1 ++ l2) = rev l2 ++ rev l1.
Proof.
  intros.
  induction l1 as [ | h1 t1 IH1].
  - simpl. Search app. rewrite -> app_nil_r. reflexivity.
  - simpl. rewrite -> IH1. rewrite <- app_assoc. reflexivity.
  (* FILL IN HERE *) Admitted.
```

```
Theorem rev_involutive : forall X : Type, forall l : list X,
  rev (rev l) = l.
Proof.
  intros.
  induction l as [ | h t IH].
  - simpl. reflexivity.
  -  simpl.  rewrite -> rev_app_distr. simpl. rewrite -> IH. reflexivity.
  (* FILL IN HERE *) Admitted.
```

分析：

第三题主要是对l1作induction，然后分别用前面已经构造出的函数作rewrite再化简即可。

第四题用了第三题所证明的定理。