1.

分析：

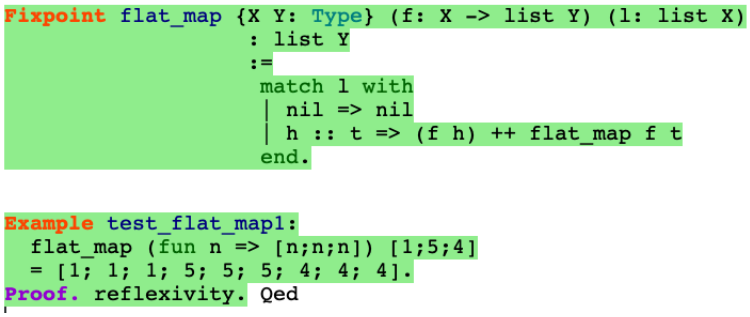
通过递归对链表中的每个元素进行处理，处理方式是通过f将其映射成一个新列表，然后用append函数将其与对子结构的递归结果拼接起来。

代码：

```
Fixpoint flat_map {X Y: Type} (f: X -> list Y) (l: list X)
                    : list Y
                    :=
                     match l with
                     | nil => nil
                     | h :: t => (f h) ++ flat_map f t
                     end.


Example test_flat_map1:
  flat_map (fun n => [n;n;n]) [1;5;4]
  = [1; 1; 1; 5; 5; 5; 4; 4; 4].
Proof. reflexivity. Qed.
```

运行结果：



2.

分析：

定义一个函数，如果是奇数则返回三倍其值，偶数则两倍。然后用map函数，将这个函数作为映射，对链表进行处理。

代码：

```
Definition func (x : nat) : nat :=
  if even x then 2 * x
  else 3 * x.


Definition changelist (l : list nat) : list nat :=
  map func l.

Example test: changelist [1;2;3;4;5;6] = [3;4;9;8;15;12].
Proof. reflexivity. Qed.
```

运行结果：

```
Definition func (x : nat) : nat :=
  if even x then 2 * x
  else 3 * x.
..
..
Definition changelist (l : list nat) : list nat :=
  map func l.

Example test: changelist [1;2;3;4;5;6] = [3;4;9;8;15;12].
Proof. reflexivity. Qed.
```

3.

分析：

定义了两个函数，分别判断x是奇数还是偶数，f1:若是奇数则返回一个封装的函数`plus 0`，否则返回函数`plus x`，f2与之相反。再利用fold，初始值设为0，就可以满足题目条件了。最后用pair将二者组合起来即可。

代码：

```
Definition f1 (x : nat) : nat -> nat :=
  if even x then plus 0
  else plus x.

Definition f2 (x : nat) : nat -> nat :=
  if even x then plus x
  else plus 0.

Definition sumPair (l : list nat) : prod nat nat :=
  pair (fold f1 l 0) (fold f2 l 0).

Example test_sumPair: sumPair [1;2;3;4;5] = (9, 6).
Proof. reflexivity. Qed.
```

运行结果：

```
Definition f1 (x : nat) : nat -> nat :=
  if even x then plus 0
  else plus x.

Definition f2 (x : nat) : nat -> nat :=
  if even x then plus x
  else plus 0.

Definition sumPair (l : list nat) : prod nat nat :=
  pair (fold f1 l 0) (fold f2 l 0).

Example test_sumPair: sumPair [1;2;3;4;5] = (9, 6).
Proof. reflexivity. Qed.
```

4.

分析：

思路还是利用fold，但是要注意几点。首先，题干中的最大值相当于告诉你了一个从0到x的集合作为初始值，然后依次取并集；其次，以链表为元素的链表的表示方法。

代码：

```
Definition bag := list nat.

Fixpoint count (v : nat) (s : bag) : nat :=
  match s with
  | nil => 0
  | h :: t => match (eqb v h) with
                | true => S (count v t)
                | false => count v t
              end
  end.

Fixpoint inter (s1: bag) (s2: bag) : bag :=
  match s1 with
  | nil => nil
  | h :: t => if ((count h s2) =? 0) then inter t s2
              else h :: inter t s2
  end.

Fixpoint ntoz (x : nat) : list nat :=
  match x with
  | O => [0]
  |S x' => [x] ++ ntoz x'
  end.

Definition bigInter (lol : list (list nat)) (x : nat) : list nat:=
  fold inter lol (ntoz x).
```

```
Example test_bigInter : bigInter [[1;3;5];[2;3;7;6;5];[3;9;8;5]] 10 = [3;5].
Proof. reflexivity. Qed.
```

运行结果：

```
Definition bag := list nat.

Fixpoint count (v : nat) (s : bag) : nat :=
  match s with
  | nil => 0
  | h :: t => match (eqb v h) with
                  | true => S (count v t)
                  | false => count v t
                end
  end.

Fixpoint inter (s1: bag) (s2: bag) : bag :=
  match s1 with
  | nil => nil
  | h :: t => if ((count h s2) =? 0) then inter t s2
                else h :: inter t s2
  end.

Fixpoint ntoz (x : nat) : list nat :=
  match x with
  | O => [0]
  |S x' => [x] ++ ntoz x'
  end.
Definition bigInter (lol : list (list nat)) (x : nat) : list nat:=
  fold inter lol (ntoz x).

Example test_bigInter : bigInter [[1;3;5];[2;3;7;6;5];[3;9;8;5]] 10 = [3;5].
Proof. reflexivity. Qed.
```