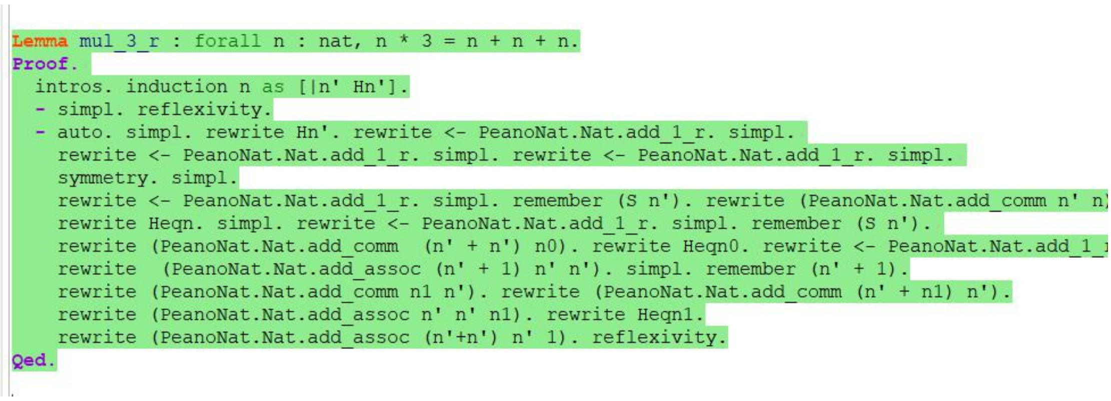
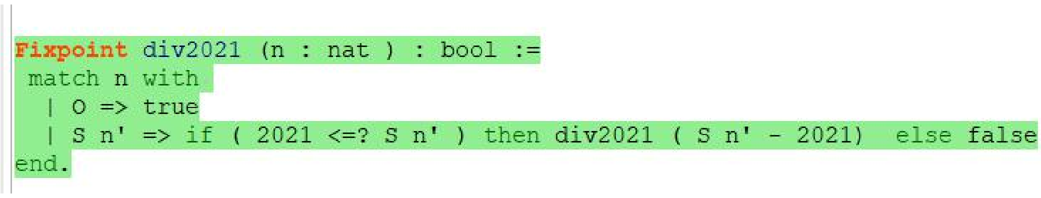
1.



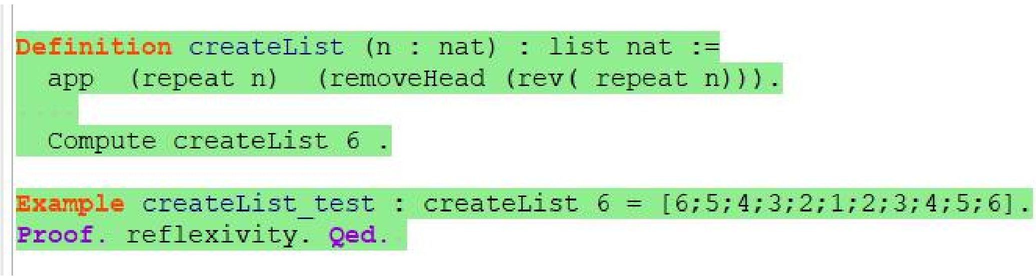
```
Lemma mul_3_r : forall n : nat, n * 3 = n + n + n.
Proof.
  intros. induction n as [|n' Hn'].
  - simpl. reflexivity.
  - auto. simpl. rewrite Hn'. rewrite <- PeanoNat.Nat.add_1_r. simpl.
    rewrite <- PeanoNat.Nat.add_1_r. simpl. rewrite <- PeanoNat.Nat.add_1_r. simpl.
    symmetry. simpl.
    rewrite <- PeanoNat.Nat.add_1_r. simpl. remember (S n'). rewrite (PeanoNat.Nat.add_comm n' n
    rewrite Heqn. simpl. rewrite <- PeanoNat.Nat.add_1_r. simpl. remember (S n').
    rewrite (PeanoNat.Nat.add_comm  (n' + n') n0). rewrite Heqn0. rewrite <- PeanoNat.Nat.add_1_
    rewrite  (PeanoNat.Nat.add_assoc (n' + 1) n' n'). simpl. remember (n' + 1).
    rewrite (PeanoNat.Nat.add_comm n1 n'). rewrite (PeanoNat.Nat.add_comm (n' + n1) n').
    rewrite (PeanoNat.Nat.add_assoc n' n' n1). rewrite Heqn1.
    rewrite (PeanoNat.Nat.add_assoc (n'+n') n' 1). reflexivity.
Qed.
```

2.



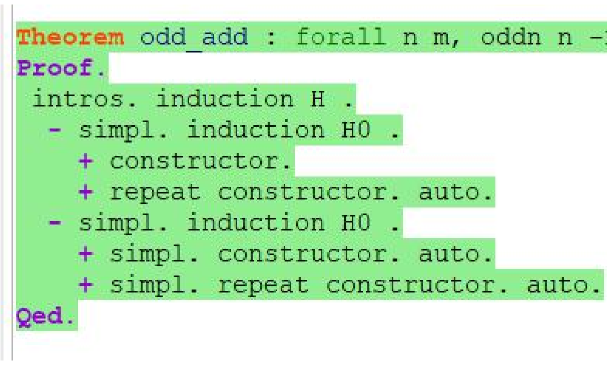
```
Fixpoint div2021 (n : nat ) : bool :=
 match n with
  | O => true
  | S n' => if ( 2021 <=? S n' ) then div2021 ( S n' - 2021)  else false
end.
```

3.



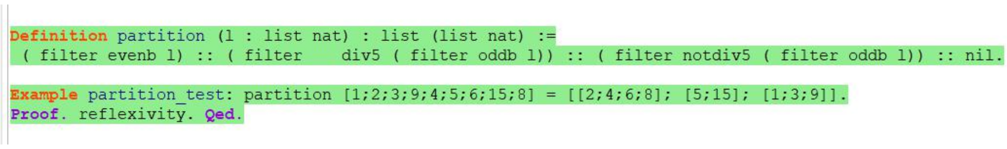
```
Definition createList (n : nat) : list nat :=
  app  (repeat n)  (removeHead (rev( repeat n))).

  Compute createList 6 .

Example createList_test : createList 6 = [6;5;4;3;2;1;2;3;4;5;6].
Proof. reflexivity. Qed.
```

4.



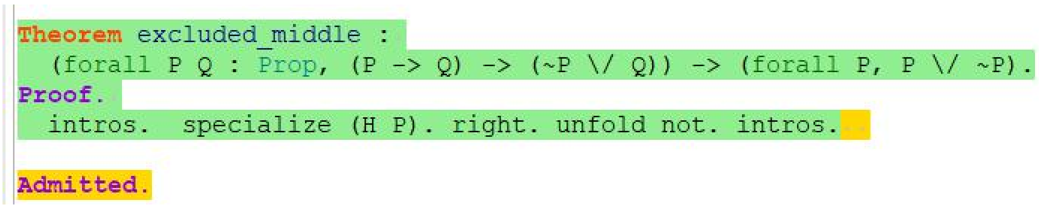
```
Theorem odd_add : forall n m, oddn n -> evenn m -> oddn (n + m).
Proof.
 intros. induction H .
  - simpl. induction H0 .
    + constructor.
    + repeat constructor. auto.
  - simpl. induction H0 .
    + simpl. constructor. auto.
    + simpl. repeat constructor. auto.
Qed.
```

5.



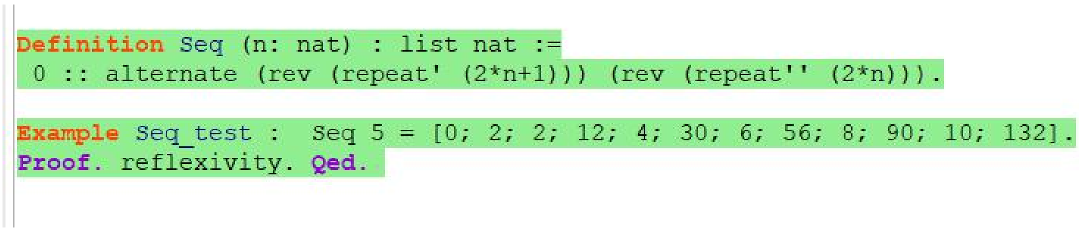
```
Definition partition (l : list nat) : list (list nat) :=
 ( filter evenb l) :: ( filter    div5 ( filter oddb l)) :: ( filter notdiv5 ( filter oddb l)) :: nil.

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

6.



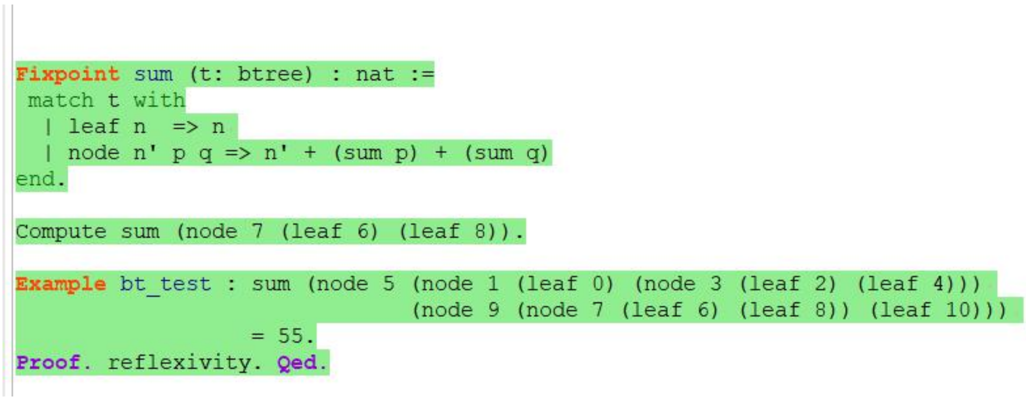
```
Theorem excluded_middle :
  (forall P Q : Prop, (P -> Q) -> (~P \/ Q)) -> (forall P, P \/ ~P).
Proof.
  intros.  specialize (H P). right. unfold not. intros.

Admitted.
```

7.



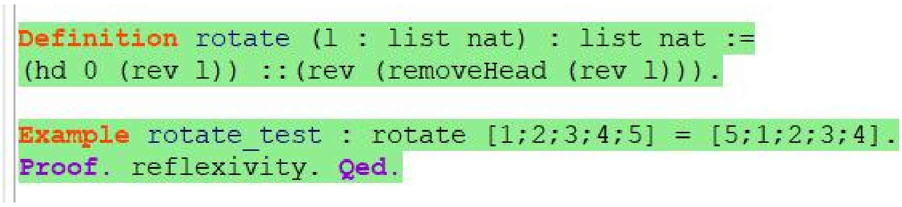
```
Definition Seq (n: nat) : list nat :=
 0 :: alternate (rev (repeat' (2*n+1))) (rev (repeat'' (2*n))).

Example Seq_test :  Seq 5 = [0; 2; 2; 12; 4; 30; 6; 56; 8; 90; 10; 132].
Proof. reflexivity. Qed.
```

8.



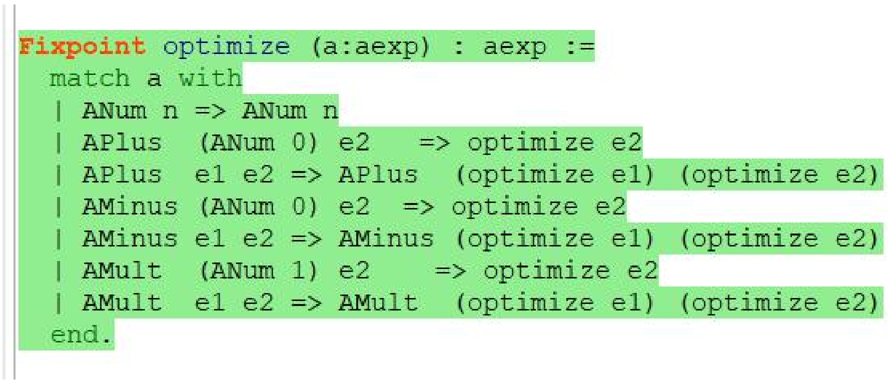
```
Fixpoint sum (t: btree) : nat :=
 match t with
   | leaf n  => n
   | node n' p q => n' + (sum p) + (sum q)
end.

Compute sum (node 7 (leaf 6) (leaf 8)).

Example bt_test : sum (node 5 (node 1 (leaf 0) (node 3 (leaf 2) (leaf 4)))
                              (node 9 (node 7 (leaf 6) (leaf 8)) (leaf 10)))
                  = 55.
Proof. reflexivity. Qed.
```

9.



```
Definition rotate (l : list nat) : list nat :=
(hd 0 (rev l)) ::(rev (removeHead (rev l))).

Example rotate_test : rotate [1;2;3;4;5] = [5;1;2;3;4].
Proof. reflexivity. Qed.
```

10.



```
Fixpoint optimize (a:aexp) : aexp :=
  match a with
   | ANum n => ANum n
   | APlus  (ANum 0) e2    => optimize e2
   | APlus  e1 e2 => APlus  (optimize e1) (optimize e2)
   | AMinus (ANum 0) e2  => optimize e2
   | AMinus e1 e2 => AMinus (optimize e1) (optimize e2)
   | AMult  (ANum 1) e2    => optimize e2
   | AMult  e1 e2 => AMult  (optimize e1) (optimize e2)
  end.
```