Isabelle/HOL Exercises
Lists

Searching in Lists

Define a function *first_pos* that computes the index of the first element in a list that satisfies a given predicate:

```
first_pos :: "('a ⇒ bool) ⇒ 'a list ⇒ nat"
```

The smallest index is *0*. If no element in the list satisfies the predicate, the behaviour of *first_pos* should be as described below.

```
primrec
  "first_pos P [] = 0"
  "first_pos P (x # xs) = (if P x then 0 else Suc (first_pos P xs))"
```

Verify your definition by computing

- the index of the first number equal to *3* in the list *[1::nat, 3, 5, 3, 1]*,
- the index of the first number greater than *4* in the list *[1::nat, 3, 5, 7]*,
- the index of the first list with more than one element in the list *[[], [1, 2], [3]]*.

Note: Isabelle does not know the operators > and ≥. Use < and ≤ instead.

```
lemma "first_pos (λ x. x = 3) [1::nat, 3, 5, 3, 1] = 1"
  by auto

lemma "first_pos (λ x. 4 < x) [1::nat, 3, 5, 7] = 2"
  by auto

lemma "first_pos (λ x. 1 < length x) [[], [1, 2], [3]] = 1"
  by auto
```

Prove that *first_pos* returns the length of the list if and only if no element in the list satisfies the given predicate.

```
lemma "list_all (λ x. ¬ P x) xs = (first_pos P xs = length xs)"
  apply (induct xs)
  apply auto
done
```

Now prove:

```
lemma "list_all (λ x. ¬ P x) (take (first_pos P xs) xs)"
  apply (induct xs)
  apply auto
done
```

How can *first_pos (λ x. P x ∨ Q x) xs* be computed from *first_pos P xs* and *first_pos Q xs*? Can something similar be said for the conjunction of *P* and *Q*? Prove your statement(s).

```
lemma "first_pos (λ x. P x ∨ Q x) xs = min (first_pos P xs) (first_pos Q xs)"
  apply (induct xs)
  apply auto
done
```

For ∧, only a lower bound can be given.

```
lemma "max (first_pos P xs) (first_pos Q xs) ≤ first_pos (λ x. P x ∧ Q x) xs"
  apply (induct xs)
  apply auto
done
```

Suppose *P* implies *Q*. What can be said about the relation between *first_pos P xs* and *first_pos Q xs*? Prove your statement.

```
lemma "(∀x. P x ⟶ Q x) ⟶ first_pos Q xs ≤ first_pos P xs"
  apply (induct xs)
  apply auto
done
```

Define a function *count* that counts the number of elements in a list that satisfy a given predicate.

```
  count :: "('a ⇒ bool) ⇒ 'a list ⇒ nat"

primrec
  "count P [] = 0"
  "count P (x # xs) = (if P x then Suc (count P xs) else (count P xs))"
```

Show: The number of elements with a given property stays the same when one reverses a list with *rev*. The proof will require a lemma.

```
lemma count_append[simp]: "count P (xs @ ys) = count P xs + count P ys"
  apply (induct xs)
  apply auto
done
```

lemma *"count P xs = count P (rev xs)"*
  apply *(induct xs)*
  apply *auto*
done

Find and prove a connection between the two functions `filter` and `count`.

lemma *"length (filter P xs) = count P xs"*
  apply *(induct xs)*
  apply *auto*
done