Exercises

- Write a function that takes as argument a list of integers, as well as a function of type `int->bool` and returns the number of elements of the list for which the function evaluates to `true`
- Test if two ordered lists are equal
- Test if a list is a palindrome

Solutions

- Write a function that takes as argument a list of integers, as well as a function of type `int->bool` and returns the number of elements of the list for which the function evaluates to `true`

```
val test = fn f =>fn l => List.length (List.filter f l);
val greatertwo = fn n => n>2;
test greatertwo a;
```

- Test if two ordered lists are equal

```
val eq = fn l1 => fn l2 => l1=l2;
eq a a;
eq a b;
```

- Test if a list is a palindrome

```
val palindrome = fn l => l = List.rev l;
palindrome a;
palindrome [1,2,3,2,1];
```

Exercise

What is the type of the following expression?

```
List.map print string;
```

Solution

```
List.map : ('a -> 'b) -> 'a list -> 'b list

print string : string -> unit

'a = string, 'b = unit

fun : string list -> unit list
```

Exercise

What is the type of the following expression

```
let f1 l = List.map (fun x y -> x,y) l;
```

Solution

```
List.map:('a -> 'b) -> 'a list -> 'b list

fun x y -> x,y :'a -> 'b -> 'a * 'b


f1:'a list->('b -> 'a * 'b) list
```

Exercise

What is the type of the following expression

```
(fun x -> x) 12;
```

Solution

```
int: 12
```

Exercise

What is the type of the following expression

```
fun x y -> x y;
```

Solution

```
x :'a -> 'b,
y :'a

fun: ('a->'b)->'a->'b
```

Exercise

What is the type of the following expression

```
let f1 x = [x ; x];
```

Solution

```
x:'a
[x; x]:'a list

f1: 'a -> 'a list
```

Exercise

What is the type of the following expression

```
let f2 x y = (x @ (y x));
```

Solution

```
(@):'a list -> 'a list -> 'a list
x:'a list
(y x):'a list
y:'a list -> 'a list

fun:'a list -> ('a list -> 'a list) -> 'a list
```

Exercise

What is the type of the following expression

```
let f3 x = List.map x;
```

Solution

```
List.map:('a -> 'b) -> 'a list -> 'b list
x:'a -> 'b
List.map x:'a list -> 'b list

f3:('a -> 'b) -> 'a list -> 'b list
```

Note that this is the same as `List.map`

Exercise

Define a function `copy: int * 'a -> 'a list` such that, for example

```
copy(0,5) = []
copy(1,5) = [5]
copy(3,"a") = ["a","a","a"]
copy(3,copy(1,8)) = [[8],[8],[8]]
```

Solution

```
fun copy(0,x) = []
   | copy(n,x) = x::copy(n-1,x);
```

Exercise

Define a function `sumlists: int list * int list -> int list` which takes two lists of integers and returns the list of the sums of the elements in corresponding position in the input lists, extending the shortest list, if needed, with 0's

Solution

```
fun sumlists(l,[]) = l
     | sumlists([],k) = k
     | sumlists(x::l,y::k) = (x+y)::sumlists(l,k);

sumlists([1,2],[3,4]);
sumlists([1],[3,4,2]);
sumlists([1,6],[3]);
```

Equality types

- Equality type: Type for which equality is defined
- Examples
  - Equality type
    - `int`, `char`, `bool`
    - `'a list`, when `'a` is an equality type
  - Not equality types
    - `real`
    - Function types

Notation

- `'a`: Arbitrary type
- `''a`: Arbitrary *equality* type

Exercise

- Define a function `remove_dup: ''a list -> ''a list` which removes duplicates from a list
- Can you define this with a more general type, `remove_dup: 'a list -> 'a list`?

Solution

```
fun delete(x,[]) = []
 | delete(x,y::l) = if x=y then delete(x,l) else y::delete(x,l);

fun remove_dup [] = []
 | remove_dup (x::l) = x::remove_dup(delete(x,l));


remove_dup [];
remove_dup [1,2,1];
remove_dup ["a","a","a"];
remove_dup [[1],[1,2],[1,2,3],[1,2],[4,5]];

remove_dup [1.1,2.1,1.1];
```

Exercise

Define a function `first_list: ('a * 'b) list -> 'a list` which returns the list consisting of the first elements only.

Example

```
first_list [(1,"a"),(2,"b"),(3,"c")] = [1,2,3]
```

Solution

```
fun first_list [] = []
  | first_list((x,y)::l) = x::first_list l;


first_list [] ;
first_list [(1,2),(1,3)];
first_list [(1,"a"),(2,"b"),(3,"c")];
first_list [([],"a"),([1],"b"),([1,2],"c")];
```