Exercise

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`

Solution

```
val rec test = fn
   empty => (fn f => 0)
  | cons (e,l) => (fn f => (if f e then 1 else 0) + test l f );
```

Exercise

Test if a list is a palindrome

Suggestion

- Find ith member of list (numbering from 0)
- Use this to define palindrome recursively

Solution

- Find ith member of list (numbering from 0)

```
val rec member= fn
  empty => (fn i => 0)
| cons(a,l)=> fn  0=>a
                |i  => member l (i-1);
```

- Last element of list

```
val rec pal = fn
    empty => (fn i=>true)
| cons(a,l) => (fn i=> (
           if i<=1 then true
           else a=member l (i-1)) andalso pal l (i-2));

val palindrome = fn
   l => pal l ((length l)-1);
```

Type checking

```
fun comb (n,m) =
  if m=0 orelse m=n then 1
  else comb(n-1,m)+comb(n-1,m-1);
```

- One branch of `if` is integer 1, so the second must be. Therefore `comb` returns an integer
- `n-1` and `m-1` mean the arguments must also be integers

Exercise

What are is the type of `foo` and why?

```
fun foo (a,b,c,d) =
   if a=b then c+1 else
     if a>b then c else c+d
```

Folding

- Summing a list, with an accumulator

```
fun sum' (acc:int) (l:int list):int =
  case l of
    [] => acc
  | x::xs => sum' (acc+x) xs;
```

- Concatenating strings, with an accumulator

```
fun concat' (acc:string) (l:string list):string =
  case l of
    [] => acc
  | x::xs => concat' (acc^x) xs;
```

Folding

- With `foldl`

```
fun sum (l:int list):int = foldl (fn (x,acc) => acc+x) 0 l;
fun concat (l:string list):string = foldl (fn (x,acc) => acc^x) "" l;
```

  Notice the inline definition of (anonymous) functions

- If `f` is a function of type `'a*'b->'b`, the expression `foldl f b[x1,x2,...,xn]` evaluates to `f(xn, f(..., f(x2, f(x1,b))))`
- `foldr` traverses the list right to left, evaluating to `f(x1,f (x2,f(...,f(xn,b))))`

Folding

- A more natural definition of concatenation

```
fun concat (l:string list):string = foldr (fn (x,acc) => x^acc) "" l;
```

- Using currying, we can omit the list argument

```
val sum = foldl (fn (x,a) => x+a) 0;
val concat = foldr (fn (x,a) => x^a) "";
```

Infix notation

- Infix binary operators: Can be made into functions by qualifying their name with the structure they belong to
- `Int.+` is therefore a function that takes in two integers and adds them
- We can therefore write

```
val sum = foldl Int.+ 0;
val concat = foldr String.^ "";
```

Folding

- Many list functions can be expressed with folding

```
fun length l = foldl (fn (_,a) => a+1) 0 l;
fun rev l = foldl List.:: [] l;
fun map f l = foldr (fn (x,a) => (f x)::a) [] l;
fun app f l = foldr (fn (x,_) => f x) () l;
fun filter f l = foldr (fn(x,a) => if f x then x::a else a) [] l;
```

More list operators

- Exists

```
val a = [1,2,3,4];
List.exists (fn a => a<2) a;
List.exists (fn a => a<1) a;
```

- All

```
List.all (fn a => a<5) a;
List.all (fn a => a<2) a;
```

Tabulate

```
List.tabulate (5,(fn n => n*n));
List.tabulate (0,(fn n => n*n));
List.tabulate (~1,(fn n => n*n));
```

Exercises

- Test whether a list is ordered
- Write a function `sum` that sums a list of integers.

Solutions

- Test whether a list is ordered

```
val a = [1,2,3,4];
val b = [1,2,4,3];
val rec ordered = fn l =>
    if List.null l then true
  else if List.null (tl l) then true
  else (hd l < hd (tl l)) andalso  ordered (tl l);
```

- Write a function `sum` that sums a list of integers.

```
val sum = fn l => foldl ( fn (n,m)  => n+m) 0 l ;
```