Q 2.1)

g1 part: The type of g1 is:

('a -> bool) -> 'a list -> 'a list

A brief argument (that is not required):
From the first clause we see p: 'a -> bool
and x: 'a (from the 'when p x' part).
Then x::xs : 'a list and x::g1 p xs : 'a list.

g1 p xs gives the longest prefix of xs where every element x_i satisfies p, i.e. p x_i = true.

g2 part: The type of g2 is:

('a -> 'a) -> ('a -> 'a) -> int -> 'a -> 'a
(f) (h) (n) (x)

Brief (not required) argument:
The match clause shows directly that n: int.
Since x and f x both occur as last argument of g2, we have f: 'a -> 'a.
f and h exchange position in calls of g2 so h: 'a -> 'a.

g2 f h n x =

fails with an exception if n < 0
f(h(f(...f x)...))
if n > 0, n is odd and there are $\frac{n+1}{2}$ applications of f and $\frac{n-1}{2}$ of h
h(f(...h(f(x))...))
if n > 0, n is even and there are $\frac{n}{2}$ applications of f and of h

Q 2.2)

```
let rec glA p xs acc =
    match xs with
    | x:: rest when p x -> glA p rest (x:: acc)
    | _                 -> List.rev acc

let gl p xs = glA p xs []

let rec glC p xs k =
    match xs with
    | x:: rest when p x
                -> glC p rest (fun v -> k(x::v))
    |           -> k []

let gl p xs = glC p xs id
```

Q 2.3)

g2 is tail recursive because the only recursive call of g2 in the last clause

```
| n -> g2 n f (n-1) (f x)
```

is in a tail position. When that call returns a value, there is no work left to be done.