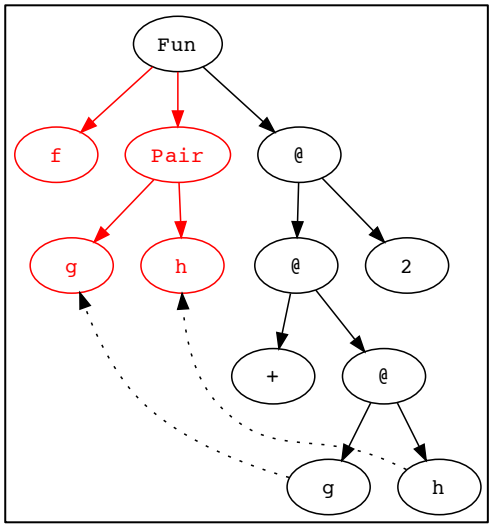
Principles of Programming Languages [PLP]
Exercises on functional programming and Haskell

1) Use the parse graph to the right to compute the most general type for the function
   `f(g,h) = g(h) + 2`
   Assume that **2** has type **Integer** and **+** has type **Integer → Integer → Integer**.



2) Suppose that the following Haskell definitions have been loaded:

```
my_const c x = c
append [] ys = ys
append (x:xs) ys = x : append xs ys
my_map f [] = []
my_map f (x:xs) = f x : my_map f xs
```

What is the type of each of the following Haskell expressions? (Some may give an error.)

```
a. my_const
b. my_const True
c. append []
d. append [True,False]
e. append [3] ['a', 'b']
f. append "quad" ['a', 'b']
g. my_map
h. my_map (my_const True)
```

What is the value of each of the following Haskell expressions?

```
a. my_const 5 "octopus"
b. my_map (my_const "squid") [1 ..]
c. my_map sqrt [1, 2, 100]
```

3) Consider the following definitions in Haskell:

```
foo x = x:(foo x)
bar x y = if (length x < 3) then (sum x) else (sum y)
```

a. Infer the type of the definitions of functions **foo** and **bar**, including type constraints.
b. What is the result of evaluating **bar [1,2] (foo 1)**?
c. And what is the result of evaluating **bar [1,2,3] (foo 1)**?

4) Consider the following tail-recursive function, written in Haskell:

```
mkMin x y  = if  x <= y then x else mkMin (x – y) y
```

a. Write the type inferred for function **mkMin** including the type constraints.
b. Assuming that the language also includes assignments and a *while* statement, transform **mkMin** into an equivalent non-recursive function.
c. Assuming that the language is pure functional and includes lambda-abstraction, transform **mkMin** into a function in Continuation Passing Style (CPS).
d. Infer the Haskell type of the latter function.

5) By exploiting the syntax for *list comprehension* of Haskell, write expressions that denote:
a. The list of squares of even natural numbers from 0 to 100;
b. The list of all *Pythagorean triples* up to *n*, i.e., of all triples *(x, y, z)* such that *x, y, z* <= *n* and x*x + y*y = z*z.

6) Using list comprehension, we can denote the list of all even numbers from 0 to 10 as
```
[2 * x | x <- [0..5]]
```
but also as
```
[2 * x | x <- [0..], x < 6]
```
Is there any difference?

7) Infer the type of the following Haskell functions (remember that "**++**" is the operator of list concatenation):

```
twice x = [x,x]
repl x y = [x..y]
f g h = \x -> \y -> (g x)++(h y)
```

What is the result evaluating the expression **f twice (repl 1) 10 5**?

8) Consider the following function:

```
int foo (int x){
     if (x>100)  return x-10;
     else  return foo(foo(x+11));
}
```
Is this tail recursive? Justify your answer.