{- -- continuação do slide anterior -- -}

```
removeTable _ Empty = Empty
removeTable x (Node (c,_) e Empty) | x == c = e
removeTable x (Node (c,_) Empty d) | x == c = d
removeTable x (Node (c,v) e d)
                        | x < c  = Node (c,v) (removeTable x e) d
                        | x > c  = Node (c,v) e (removeTable x d)
                        | x == c = let (y,z) = minTable d
                                   in Node (y,z) e (removeTable y d)


minTable :: Table a b  -> (a,b)
minTable (Node (c,v) Empty _) = (c,v)
minTable (Node _ e _)         = minTable e


instance (Show a,Show b) => Show (Table a b) where
    show Empty = ""
    show (Node (c,v) e d) = (show e)++(show c)++"\t"++(show v)++"\n"++(show d)
```



```
module Main where

import Table

type Numero = Integer
type Nome   = String
type Nota   = Integer

pauta :: [(Numero,Nome,Nota)] -> Table Numero (Nome,Nota)
pauta [] = newTable
pauta ((x,y,z):xyzs) = updateTable (x,(y,z)) (pauta xyzs)

info = [(1111,"Mario",14), (5555,"Helena",15), (3333,"Teresa",12),
        (7777,"Pedro",15), (2222,"Rui",17), (9999,"Pedro",10)]
```

Exemplos:

```
*Main> pauta info
1111   ("Mario",14)
2222   ("Rui",17)
3333   ("Teresa",12)
5555   ("Helena",15)
7777   ("Pedro",15)
9999   ("Pedro",10)
```

```
*Main> findTable 5555 (pauta info)
Just ("Helena",15)
*Main> findTable 8888 (pauta info)
Nothing
*Main> removeTable 9999 (pauta info)
1111   ("Mario",14)
2222   ("Rui",17)
3333   ("Teresa",12)
5555   ("Helena",15)
7777   ("Pedro",15)
```

Como estará a tabela implementada ?

Sequência de Fibonacci

O n-ésimo número da sequência de Fibonacci define-se matematicamente por

$$\begin{aligned} fib\ n &= 0 && ,\ se\ \ n = 0 \\ fib\ n &= 1 && ,\ se\ \ n = 1 \\ fib\ n &= fib\ (n\text{-}2) + fib\ (n\text{-}1) && ,\ se\ \ n \geq 2 \end{aligned}$$

```
fib 0 = 0
fib 1 = 1
fib n | n>=2  = fib (n-2) + fib (n-1)
```

O cálculo do fib de um número pode envolver o cálculo do fib de números mais pequenos, repetidas vezes.

```
fib 5 ⇒  (fib 3)+(fib 4)  ⇒  ((fib 1)+(fib 2))+((fib 2)+(fib 3))
 ⇒ (1+((fib 0)+(fib 1)))+((fib 2)+(fib 3)) ⇒  ... ⇒  ... ⇒ 5
```

A sequência de Fibonnacci pode ser definida por

```
seqFibonnacci = [ fib n | n <- [0,1..]  ]
```


Uma versão mais eficiente dos números de Fibonnacci utiliza um parametro de acumulação.

Neste caso o acumulador é um par que regista os dois últimos números de Fibonnacci calculados até ao momento.

```
fib n = fibAc (0,1) n
  where fibAc (a,b) 0 = a
        fibAc (a,b) 1 = b
        fibAc (a,b) (n+1) = fib (b,a+b) n
```

```
fib 5  ⇒  fibAc (0,1) 5  ⇒  fibAc (1,1) 4  ⇒  fibAc (1,2) 3
       ⇒  fibAc (2,3) 2  ⇒  fibAc (3,5) 1  ⇒  5
```

A sequência de Fibonnacci pode ser definida por

```
seqFib = 0 : 1 : [ a+b | (a,b) <- zip seqFib (tail seqFib) ]
```

Note que é a lazy evaluation que faz com que este género de definição seja possível.