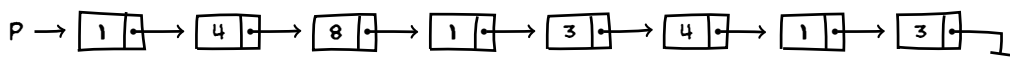
Removendo todas as cópias do k

Imagine que o ponteiro p aponta para uma lista encadeada que pode conter elementos repetidos



A tarefa é

→ *remover todas as cópias do número* k

Bom, dessa vez nós vamos fazer as coisas na volta.

Quer dizer, as chamadas recursivas vão até o Nulo.

E na volta, nós removemos o elemento sempre que ele é igual a k.

A coisa fica assim

```
func RTCK-Rec (q,k)                                  // Remove Todas as Cópias do k

      Se ( q == Nulo )   Retorna (q)
      Senão
            resp = RTCK-Rec (q->prox,k)              // vai até o fim

            Se (q->num != k)
                  q->prox = resp
                  Retorna (q)
            Senão
                  free (q)                           // removendo o k na volta
                  Retorna (resp)                     //
```

Legal.

Mas essa não é a única maneira de fazer as coisas.

Quer dizer, a gente também pode retornar uma lista com todos os elementos que foram removidos para a função `main()`.

A coisa fica assim

```
func RTCK2-Rec (q,k)                                 // Remove Todas as Cópias do k

      Se ( q == Nulo )   Retorna (Nulo,Nulo)
      Senão
            px,r = RTCK2-Rec (q->prox,k)             // vai até o fim

            Se (q->num != k)
                  q->prox = px                       // truque da re-conexão
                  Retorna (q,r)
            Senão
                  q->prox = r                        // insere o k na lista de removidos
                  Retorna (px,q)                     //
```

E a função `main()` fica assim

```
func main ()

      p,r = RTCK2-Rec (p,1)
      aux = r;   cont = 0
      Enquanto (aux != Nulo)                         // conta elementos removidos
            cont++;   aux = aux->prox

      Imprime (cont, 'elementos removidos')
```