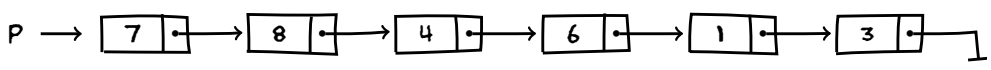
A operação de remoção

Imagine que o ponteiro `p` aponta para uma lista encadeada



A tarefa é

→ *remover o elemento* `k` *da lista*

Dessa vez, as chamadas recursivas vão até o elemento que vai ser removido — (*ou até o* `Nulo`*, caso ele não esteja lá ...*)

Daí, a chamada que vê o `k` libera o seu registro, e retorna um ponteiro para o próximo elemento.

E o resto é só o truque da reconexão.

A coisa fica assim

```
func remoção-Rec (q,k)
      Se ( q == Nulo )   Retorna (Nulo)
      Sse ( q->num == k )
            m = q->prox                          // segura o próximo elemento
            free (q)                             // e libera o registro
            Retorna (m)
      Senão
            resp = remoção-Rec (q->prox,k)
            q->prox = resp                       // re-conexão
            Retorna (q)
```

E a função `main()` fica assim

```
func main ()
      p = remoção-Rec (p,9)
```

Legal.

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

Quer dizer, em alguns casos é útil receber um ponteiro para o elemento que foi removido — (*ao invés de liberar o seu registro da memória*)

E para isso, nós vamos fazer a nossa função retornar dois valores.

A coisa fica assim

```
func remoção2-Rec (q,k)
Se ( q == Nulo )   Retorna (Nulo,Nulo)
Sse ( q->num == k )
      m = q->prox                                // segura o próximo elemento
      q->prox = Nulo                             // e desconecta o registro
      Retorna (m,q)
Senão
      px,r = remocao2-Rec (q->prox,k)
      q->prox = px                               // re-conexão
      Retorna (q,r)
```

E a função `main()` fica assim

```
func main ()
      p,r = remoção2-Rec (p,9)
      Se ( r == Nulo)
            Imprime ('o elemento não estava lá ...')
      Senão
            Imprime ('o elemento foi removido!')
```

◇