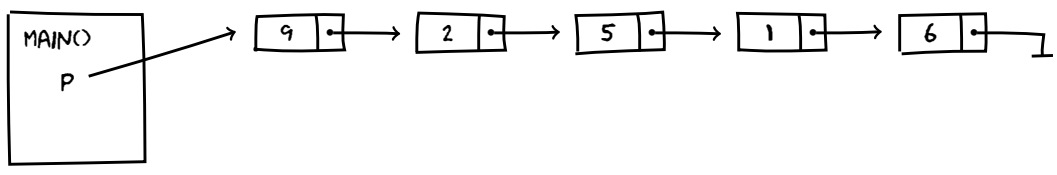
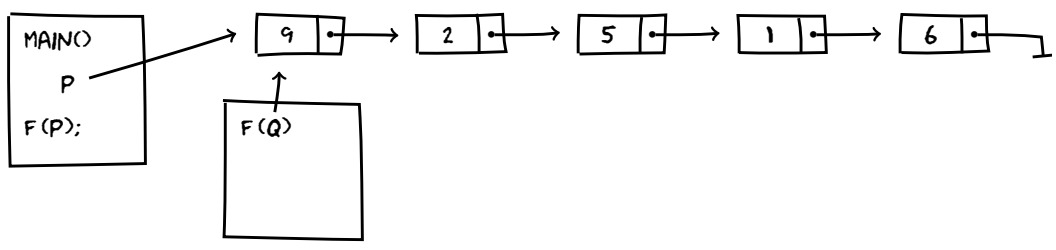
Algoritmos recursivos

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

E imagine também a função principal do seu programa

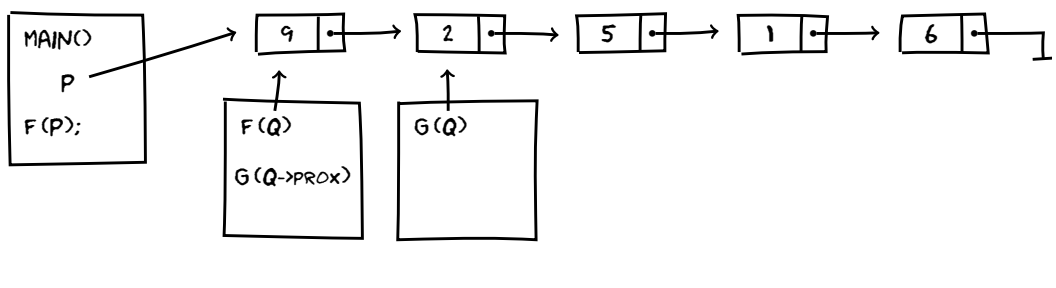


Agora, imagine que o seu programa faz a chamada a uma outra função, passando o ponteiro `p` como parâmetro

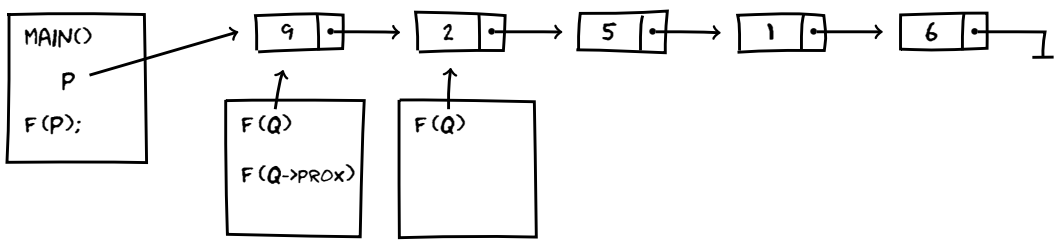


Sim! isso é como um dedo que foi colocado sobre o primeiro elemento.

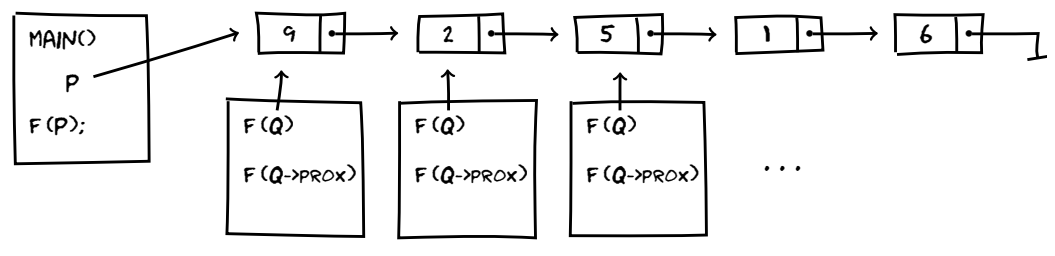
Agora, é fácil imaginar a função `F()` fazendo a chamada a uma outra função, passando `q->prox` como parâmetro



Só que, a coisa fica mais legal se a função `F()` faz uma chamada a ela mesma — (*i.e., uma chamada recursiva*)



Porque agora as chamadas vão continuar acontecendo, uma depois da outra

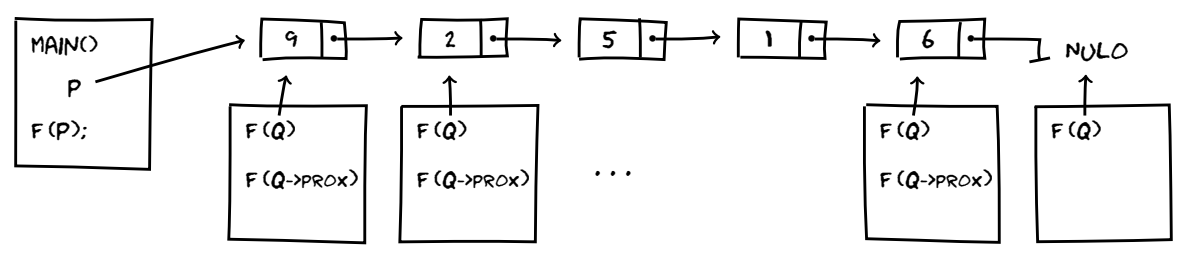


Sim! isso é como um dedo que percorre a lista.

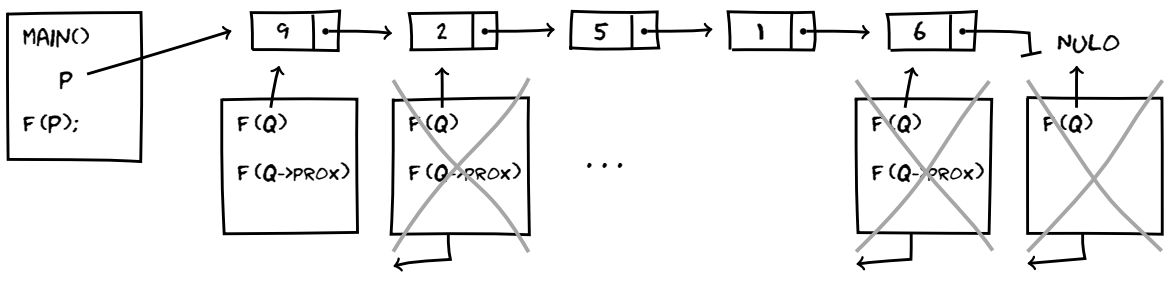
Mas, quando é que a coisa pára?

Bom, as chamadas vão continuar acontecendo até que a gente chegue no `Nulo`.

Então, é ali que a coisa pára



Daí, as chamadas vão terminando uma por uma — (*na ordem contrária em que elas aconteceram*)



Não é legal?

Agora nós temos uma outra maneira de percorrer a lista, por meio de uma função recursiva

```
func F-Rec (q)

      Se ( q == Nulo )   Retorna                  // já acabei

      F( q->prox )                                // passa para o próximo

      ( . . . )

      Retorna                                     // e volta
```

E para fazer alguma coisa de útil com isso, basta preencher os 3 pontinhos.