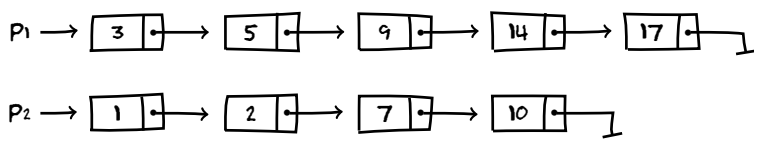
A operação de intercalação

Imagine que os ponteiros p1 e p2 apontam para duas listas encadeadas ordenadas



A tarefa é

→ *produzir uma única lista encadeada ordenada com os elementos de* p1 *e* p2

Bom, a ideia é fazer as coisas na ida.

Quer dizer, a lista vai ser construída em um parâmetro p que é passado para a função.

No início, a função `main()` atribui o valor `Nulo` a esse parâmetro.

E daí, cada chamada adiciona um elemento à lista que está sendo construída: o menor entre o 1o elemento de p1 e o 1o elemento de p2.

No momento em que uma das listas fica vazia, a outra lista é colocada no fim da lista que está sendo construída — (*e a função retorna*)

As inserções são sempre feitas no fim.

E para fazer isso de maneira eficiente, nós também utilizamos o parâmetro u que aponta para o último elemento da lista p.

A coisa fica assim

```
func intercalação-Rec (q1,q2,p,u)

      Se (q1 == Nulo)                                  // a 1a lista já acabou
            Se (p == Nulo)   Retorna (q2)
            Senão   u->prox = q2;   Retorna (p)

      Sse (q2 == Nulo)                                 // a 2a lista já acabou
            Se (p == Nulo)   Retorna (q1)
            Senão   u->prox = q1;   Retorna (p)

      Senão
            Se (q1->num < q2->num)                     //
                  m = q1;   q1 = q1->prox              // segura o menor elemento
            Senão                                      //
                  m = q2;   q2 = q2->prox              //
            m->prox = Nulo

            Se (p == Nulo)                             //
                  p = m;   u = m                       // e insere no fim
            Senão                                      //
                  u->prox = m;   u = m                 //

            resp = intercalação-Rec (q1,q2,p,u)
            Retorna (resp)
```

E a função `main()` fica assim

```
func main ()

      p = intercalação-Rec (p1,p2,Nulo,Nulo)
```

O algoritmo Mergesort

Agora que nós temos a função de intercalação (recursiva) é fácil escrever o algoritmo Mergesort (recursivo).

Mas antes disso, é preciso ter uma função que quebra a lista em duas — (*de maneira recursiva*)

Só que isso é bem fácil.

Quer dizer, nós vamos fazer a coisa na volta.

A cada passo, a chamada recursiva coloca o seu elemento em p1 ou p2 — (*de maneira alternada*)

E no final, metade dos elementos vai estar em p1, e a outra metade em p2.

A coisa fica assim

```
func divide-Rec (q)

      Se (q == Nulo)   Retorna (Nulo,Nulo)
      Senão
            p1,p2 = divide-Rec (q->prox)
            q->prox = p1;   p1 = q
            Retorna (p2,p1)                          // inverte a ordem
```

Certo.

Agora nós estamos prontos para escrever a função `mergesort-Rec()`

```
func mergesort-Rec (q)

      Se (q == Nulo)   Retorna (Nulo)                // já ordenei ...
      Senão
            q1,q2 = divide-Rec (q)
            q1 = mergesort-Rec (q1)                  // ordena 1a metade
            q2 = mergesort-Rec (q1)                  // ordena 2a metade
            q = intercalação-Rec (q1,q2,Nulo,Nulo)
            Retorna (q)
```

◇