DICAS – LISTA LIGADA

Para se percorrer uma lista ligada (ou encadeada), deve-se criar uma variável do tipo Node, que eu costumo chamar de **atual**, e fazer essa variável conter o endereço dos nós da lista, um de cada vez, partindo-se do next do head.

```
Node atual;       /* declaração da variável atual */
atual = head.getNext();     /* atual recebe o next do head, ou seja,
                               atual recebe o endereço do 1° nó */
                            /* se a lista estiver vazia, logo de cara
                               atual tem o valor null */
```

Aí, na hora em que se está percorrendo a lista, ficamos num loop enquanto atual for diferente de null.
Dentro do loop, colocamos o código para fazer o que for necessário para o nó atual da lista. Se estivermos exibindo a lista, então dentro do loop, exibimos o info do nó atual. Se estivermos buscando um determinado valor, então comparamos o info do nó atual com o valor procurado, etc.
E também devemos atualizar o atual (fazer o atual "andar").

```
while (atual != null) {
   /* faz o que for necessário para o nó atual da lista */

    /* atualiza o atual */
   atual = atual.getNext();
}
```

Por exemplo, se a lista tiver esses valores (as flechas p/ direita representam o next dos nós):

head → 2 → 7 →10

No começo, atual terá o endereço do nó com valor 2 (next do head):

head → 2 → 7 →10
↑
atual

Depois, atual receberá o endereço do nó com valor 7 (next do nó 2 ou next do atual):

head → 2 → 7 →10
↑
atual

Depois, atual receberá o endereço do nó com valor 10 (next do atual):

head → 2 → 7 → 10
↑
atual

Depois, atual receberá null (a lista acabou):

head → 2 → 7 → 10
atual = null

É dessa forma que funcionam os métodos que exibe a lista e que busca um determinado valor na lista. (métodos que fizemos em sala no projeto da ListaEncadeada).

Quando queremos inserir um nó no meio da lista, ou quando queremos remover um nó da lista, o ideal é percorremos a lista com mais uma variável do tipo Node, que costumo chamar de **ant**, que contém o endereço do nó anterior ao atual.
Então, ant começa valendo head, uma vez que atual começa valendo o endereço do 1° nó da lista.

```
Node atual;       /* declaração da variável atual */
Node ant;         /* declaração da variável ant */
atual = head.getNext();     /* atual recebe o next do head, ou seja,
                               atual recebe o endereço do 1° nó */
ant = head;               /* ant recebe o endereço do nó head */
```

No final do loop, fazemos ant receber o valor de atual, antes de atualizarmos o valor de atual, de maneira que ant sempre contém o endereço do nó anterior ao atual

Por exemplo, se a lista tiver esses valores:

head → 2 → 7 →10

No começo, atual terá o endereço do nó com valor 2, ant terá o endereço do head:

head → 2 → 7 →10
↑ ↑
ant atual

Depois, ant receberá o valor de atual, atual receberá o endereço do nó com valor 7:

head → 2 → 7 →10
↑ ↑
ant atual

Depois, ant receberá o valor de atual, atual receberá o endereço do nó com valor 10:

head → 2 → 7 → 10
↑ ↑
ant atual

Depois, ant receberá o valor de atual, atual receberá null (a lista acabou):

head → 2 → 7 → 10
↑
ant atual = null

Para inserir um nó no meio da lista, deve-se percorrer a lista, procurando-se aonde se deve inserir o nó. Se for para que a lista esteja sempre em ordem crescente, o lugar a se inserir é quando se encontra um nó atual de valor maior do que o valor a ser inserido. (Se for para que lista esteja sempre em ordem decrescente, o lugar a se inserir é quando se encontra um nó atual de valor menor do que o valor a ser inserido)

No exemplo da ilustração acima, se quisermos inserir um nó de valor 8, ele deve ser inserido entre os nós de valor 7 e 10.
Se quisermos inserir um nó de valor 15, ele deverá ser inserido no final da lista.

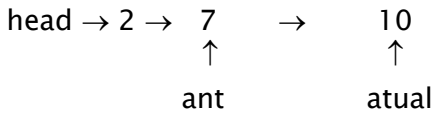
Então, temos que percorrer a lista com o loop, usando as variáveis ant e atual.
E também o ideal é usarmos uma variável booleana chamada inseriu, inicializada com false, para indicar se já inseriu ou não, quando se termina de percorrer a lista.

```
boolean inseriu = false;      /* declara uma variável booleana inseriu,
                                 inicializada com false */

/* criar um nó novo com o valor a ser inserido */

while (atual != null && !inseriu) {
     /* se for o lugar para inserir
         então {
                next do ant ← novo
                next do novo ← atual
                inseriu ← true
               }
        senão {
                ant ← atual
                atual ← next do atual
              }
      */
}  /* fim do loop */

if (!inseriu) {     /* se percorreu toda a lista e não inseriu
                       então, insere no final (ant contém o endereço
                       do último nó  */
     next do ant ← novo
}
```

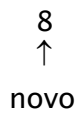
Por exemplo, se a lista tiver esses valores, e quisermos inserir o nó 8

head → 2 → 7 →10

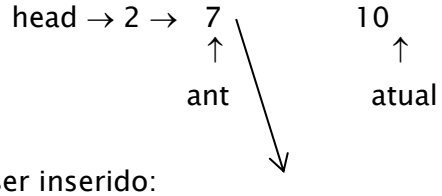
Quando encontramos o lugar para inserir o nó 8, de forma que fique em ordem crescente, será quando atual aponta para o nó de valor 10:



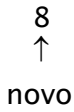
Nó a ser inserido:



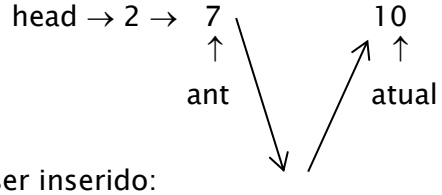
Fazemos o next do ant receber novo:



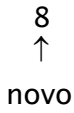
Nó a ser inserido:



E depois o next do novo receber atual:



Nó a ser inserido:



Resultado final (após a inserção):

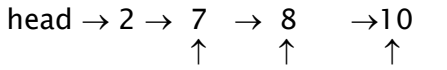
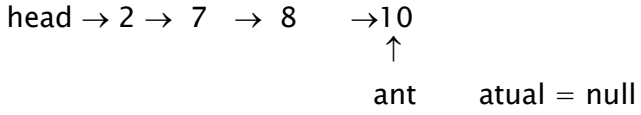
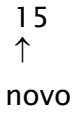


Ilustração da inserção do nó de valor 15.
A lista será percorrida e não encontraremos um nó de valor maior do que 15.
Nesse caso, o loop terminará, pois atual será null e inseriu é false (pois não inseriu no meio).



Nó a ser inserido:



Por isso, terminado o loop, devemos comparar se já inseriu. Se ainda não inseriu, como é o caso, então fazermos o next do ant receber novo: