## 10.2. Implementações recursivas

Uma lista pode ser definida de maneira recursiva. Podemos dizer que uma lista encadeada é representada por:

- uma lista vazia; ou
- um elemento seguido de uma (sub-)lista.

Neste caso, o segundo elemento da lista representa o primeiro elemento da sub-lista. Com base na definição recursiva, podemos implementar as funções de lista recursivamente. Por exemplo, a função para imprimir os elementos da lista pode ser re-escrita da forma ilustrada abaixo:

```
/* Função imprime recursiva */
void imprime_rec (Lista* l)
{
   if (vazia(l))
      return;
   /* imprime primeiro elemento */
   printf("info: %d\n",l->info);
   /* imprime sub-lista */
   imprime_rec(l->prox);
}
```

É fácil alterarmos o código acima para obtermos a impressão dos elementos da lista em ordem inversa: basta invertermos a ordem das chamadas às funções `printf` e `imprime_rec`.

A função para retirar um elemento da lista também pode ser escrita de forma recursiva. Neste caso, só retiramos um elemento se ele for o primeiro da lista (ou da sub-lista). Se o elemento que queremos retirar não for o primeiro, chamamos a função recursivamente para retirar o elemento da sub-lista.

```
/* Função retira recursiva */
Lista* retira_rec (Lista* l, int v)
{
   if (vazia(l))
      return l;   /* lista vazia: retorna valor original */

   /* verifica se elemento a ser retirado é o primeiro */
   if (l->info == v) {
      Lista* t = l;     /* temporário para poder liberar */
      l = l->prox;
      free(t);
   }
   else {
      /* retira de sub-lista */
      l->prox = retira_rec(l->prox,v);
   }
   return l;
}
```

A função para liberar uma lista também pode ser escrita recursivamente, de forma bastante simples. Nessa função, se a lista não for vazia, liberamos primeiro a sub-lista e depois liberamos a lista.

```
void libera_rec (Lista* l)
{
   if (!vazia(l))
   {
      libera_rec(l->prox);
      free(l);
   }
}
```

Exercício: Implemente uma função que verifique se duas listas encadeadas são iguais. Duas listas são consideradas iguais se têm a mesma seqüência de elementos. O protótipo da função deve ser dado por:

```
int igual (Lista* l1, Lista* l2);
```

Exercício: Implemente uma função que crie uma cópia de uma lista encadeada. O protótipo da função deve ser dado por:

```
Lista* copia (Lista* l);
```

## *10.3. Listas genéricas*

Um nó de uma lista encadeada contém basicamente duas informações: o encadeamento e a informação armazenada. Assim, a estrutura de um nó para representar uma lista de números inteiros é dada por:

```
struct lista {
   int info;
   struct lista *prox;
};
typedef struct lista Lista;
```

Analogamente, se quisermos representar uma lista de números reais, podemos definir a estrutura do nó como sendo:

```
struct lista {
   float info;
   struct lista *prox;
};
typedef struct lista Lista;
```

A informação armazenada na lista não precisa ser necessariamente um dado simples. Podemos, por exemplo, considerar a construção de uma lista para armazenar um conjunto de retângulos. Cada retângulo é definido pela base `b` e pela altura `h`. Assim, a estrutura do nó pode ser dada por:

```
struct lista {
   float b;
   float h;
   struct lista *prox;
};
typedef struct lista Lista;
```

Esta mesma composição pode ser escrita de forma mais clara se definirmos um tipo adicional que represente a informação. Podemos definir um tipo `Retangulo` e usá-lo para representar a informação armazenada na lista.

```
struct retangulo {
   float b;
   float h;
};
typedef struct retangulo Retangulo;
```