1. Escreva uma função que *conte* o número de células de uma lista encadeada. Faça duas versões: uma iterativa e uma recursiva.
2. Altura. A *altura* de uma célula c em uma lista encadeada é a distância entre c e o fim da lista. Mais precisamente, a altura de c é o número de passos do caminho que leva de c até a última célula da lista. Escreva uma função que calcule a altura de uma dada célula.
3. Profundidade. A *profundidade* de uma célula c em uma lista encadeada é o número de passos do único caminho que vai da primeira célula da lista até c. Escreva uma função que calcule a profundidade de uma dada célula.

A função abaixo promete ter o mesmo comportamento da função busca acima. Critique o código.

```
celula *busca (int x, celula *le) {

  celula *p = le;

  int achou = 0;

  while (p != NULL && !achou) {

    if (p->conteudo == x) achou = 1;

    p = p->prox; }

  if (achou) return p;

  else return NULL;

}
```

Critique o código da seguinte variante da função busca.

```
celula *busca (int x, celula *le) {

  celula *p = le;

  while (p != NULL && p->conteudo != x)

    p = p->prox;

  if (p != NULL)  return p;

  else  printf ("x não está na lista\n");

}
```

4. Escreva uma função que verifique se uma lista encadeada que contém números inteiros está em ordem crescente.
5. Escreva uma função que faça uma busca em uma lista encadeada *crescente*. Faça versões recursiva e iterativa.
6. Escreva uma função que encontre uma célula com conteúdo mínimo. Faça duas versões: uma iterativa e uma recursiva.
7. Escreva uma função que verifique se duas listas encadeadas são *iguais*, ou melhor, se têm o mesmo conteúdo. Faça duas versões: uma iterativa e uma recursiva.
8. Ponto médio. Escreva uma função que receba uma lista encadeada e devolva o endereço de uma célula que esteja o mais próximo possível do meio da lista. Faça isso sem contar explicitamente o número de células da lista.
9. Escreva versões das funções busca e busca_r para listas encadeadas com cabeça.
10. Escreva uma função que verifique se uma lista encadeada com cabeça está em ordem crescente. (Suponha que as células contêm números inteiros.)

    Por que a seguinte versão da função insere não funciona?

```
 void insere (int x, celula *p) {

   celula nova;

   nova.conteudo = x;

   nova.prox = p->prox;

   p->prox = &nova;

 }
```

11. Escreva uma função que insira uma nova célula em uma lista encadeada *sem* cabeça. (Será preciso tomar algumas decisões de projeto antes de começar a programar.)
12. Escreva uma função que faça uma *cópia* de uma lista encadeada. Faça duas versões da função: uma iterativa e uma recursiva.
13. Escreva uma função que *concatene* duas listas encadeadas (isto é, engate a segunda no fim da primeira). Faça duas versões: uma iterativa e uma recursiva.
14. Escreva uma função que insira uma nova célula com conteúdo x *imediatamente depois* da k-ésima célula de uma lista encadeada. Faça duas versões: uma iterativa e uma recursiva.
15. Escreva uma função que *troque de posição* duas células de uma mesma lista encadeada.
16. Escreva uma função que *inverta* a ordem das células de uma lista encadeada (a primeira passa a ser a última, a segunda passa a ser a penúltima etc.). Faça isso sem usar espaço auxiliar, apenas alterando ponteiros. Dê duas soluções: uma iterativa e uma recursiva.
17. Alocação de células. É uma boa ideia alocar as células de uma lista encadeada uma-a-uma? (Veja observação sobre alocação de pequenos blocos de bytes no capítulo *Alocação dinâmica de memória*.) Proponha alternativas.

Critique a seguinte versão da função remove:

```
void remove (celula *p, celula *le) {

  celula *lixo;

  lixo = p->prox;

  if (lixo->prox == NULL)  p->prox = NULL;

  else  p->prox = lixo->prox;

  free (lixo);

}
```

18. Suponha que queremos remover a primeira célula de uma lista encadeada le não vazia. Critique o seguinte fragmento de código:

```
celula **p;
p = &le;
le = le->prox;
free (*p);
```

23. Invente um jeito de remover uma célula de uma lista encadeada *sem* cabeça. (Será preciso tomar algumas decisões de projeto antes de começar a programar.)
24. Escreva uma função que *desaloque* todas as células de uma lista encadeada (ou seja, aplique a função free a todas as células). Estamos supondo que cada célula da lista foi originalmente alocado por malloc. Faça duas versões: uma iterativa e uma recursiva.
25. Problema de Josephus. Imagine uma roda de n pessoas numeradas de 1 a n no sentido horário. Começando com a pessoa de número 1, percorra a roda no sentido horário e elimine cada m-ésima pessoa enquanto a roda tiver duas ou mais pessoas. Qual o número do sobrevivente?
26. Escreva uma função que copie o conteúdo de um vetor para uma lista encadeada preservando a ordem dos elementos. Faça duas versões: uma iterativa e uma recursiva.
27. Escreva uma função que copie o conteúdo de uma lista encadeada para um vetor preservando a ordem dos elementos. Faça duas versões: uma iterativa e uma recursiva.
28. União. Digamos que uma *lesc* é uma lista encadeada sem cabeça que contém uma sequência estritamente crescente de números inteiros. (Portanto, uma lesc representa um *conjunto* de números.) Escreva uma função que faça a *união* de duas lescs produzindo uma nova lesc. A lesc resultante deve ser construída com as células das duas lescs dadas.
29. Listas encadeadas sem ponteiros. Implemente uma lista encadeada sem usar endereços e ponteiros. Use dois vetores paralelos: um vetor conteudo[0..N-1] e um vetor prox[0..N-1]. Para cada i no conjunto 0..N-1, o par (conteudo[i], prox[i])

representa uma célula da lista. A célula seguinte é (conteudo[j], prox[j]), sendo j = prox[i]. Escreva funções de busca, inserção e remoção para essa representação.

30. Esta questão trata de listas encadeadas que contêm strings ASCII (cada célula contém uma string). Escreva uma função que verifique se uma lista desse tipo está em ordem lexicográfica. As células são do seguinte tipo:
    typedef struct reg {

char *str; struct reg *prox;

    } celula;

31. ★ Contagem de palavras. Digamos que um *texto* é um vetor de bytes, todos com valor entre 32 e 126. (Cada um desses bytes representa um caractere ASCII.) Digamos que uma *palavra* é um segmento maximal de texto que consiste apenas de letras. Escreva uma função que receba um texto e imprima uma relação de todas as palavras que ocorrem no texto juntamente com o número de ocorrências de cada palavra. Use uma lista encadeada para armazenar as palavras.
32. Escreva uma função busca-e-remove para listas encadeadas *sem* cabeça.
33. Escreva uma função para remover de uma lista encadeada *todas* as células que contêm y.
34. Escreva uma função que remova a k-ésima célula de uma lista encadeada sem cabeça. Faça duas versões: uma iterativa e uma recursiva.
35. Escreva uma função busca-e-insere para listas encadeadas *sem* cabeça.
36. Descreva, em linguagem C, a estrutura de uma célula de uma lista duplamente encadeada.
37. Escreva uma função que remova de uma lista duplamente encadeada a célula apontada por p. Que dados sua função recebe? Que coisa devolve?
38. Escreva uma função que insira uma nova célula com conteúdo x em uma lista duplamente encadeada logo após a célula apontada por p. Que dados sua função recebe? Que coisa devolve?