LISTA DE EXERCÍCIOS

Endereço

1-Escreva uma função que *contém* 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 exatamente, 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.

Busca

1-Escreva uma função que verifique se uma lista encadeada que contém números inteiros está em ordem crescente.

2-Escreva uma função que faça uma busca em uma lista encadeada *crescente*. Faça versões recursiva e iterativa.

3-Escreva uma função que encontre uma célula com conteúdo mínimo. Faça duas versões: uma iterativa e uma recursiva.

4-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.

5-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.

Head Cell

1-Escreva versões das funções `busca` e `busca_r` para listas encadeadas com cabeça.

2-Escreva uma função que verifique se uma lista encadeada com cabeça está em ordem crescente. (Suponha que as células contêm números inteiros.)

INSERÇÃO

1-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.)

2-Escreva uma função que faça uma *cópia* de uma lista encadeada. Faça duas versões da função: uma iterativa e uma recursiva.

3-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.

4-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.

5-Escreva uma função que *troque de posição* duas células de uma mesma lista encadeada.

6-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.

7-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.

REMOÇÃO

1-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.)

2-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 toda célula da lista foi originalmente alocado por `malloc`. Faça duas versões: uma iterativa e uma recursiva.

3-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?