Lista de Exercícios

Considere a seguinte declaração para nós de uma lista ligada.

```
typedef struct No {
  int dado;
  struct No *prox;
} No, *p_no;
```

Considere a seguinte declaração para nós de uma árvore binária.

```
typedef struct No {
  int chave;
  struct No *esq, *dir;
} No, *p_no;
```

1. Considere a seguinte função:

   ```
   void Misterio (p_no p) {
     p_no q;

     q = p->prox;
     p->info = q->info;
     p->prox = q->prox;
     free(q);
   }
   ```

   que recebe um apontador para um nó (não necessariamente o primeiro) de uma lista ligada simples. O que ela faz? Em que condições ela pode ser aplicada?

2. Suponha que queremos manter uma lista ligada onde os nós estão ordenados em ordem crescente pelo campo dado. Implemente em linguagem C as seguintes operações:

   (a) imprime todos os dados dos nós da lista;

   (b) inicializa a lista;

   (c) verifica se a lista está vazia;

   (d) busca um nó com info i e devolve um apontador para esse (se existir, senão devolve NULL);

   (e) insere um novo nó com info i;

   (f) remove um nó com info i.

3. Refaça o exercício anterior com as seguintes variantes:

   (a) lista ligada com nó cabeça;

   (b) lista duplamente ligada;

   (c) lista duplamente ligada com nó cabeça.

4. Considere polinômios $P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ representados por uma lista ligada onde cada nó tem a forma:

```
typedef struct Pol {
  int grau;
  float coef;
  struct Pol *prox;
} Pol, *p_Pol;
```

Os nós estão ordenados em ordem crescente de grau e não há nós com campo coef igual a 0. Implemente funções para as seguintes operações:

(a) calcular a derivada de um polinômio;

(b) avaliar um polinômio em um ponto x;

(c) calcular a soma de dois polinômios;

(d) calcular o produto de dois polinômios.

Em cada operação, nenhum dos polinômios pode ser modificado. Você pode usar rotinas auxiliares.

5. Implemente uma fila por meio de uma lista ligada com nó cabeça de modo a suportar as seguintes operações:

   (a) criar uma fila (como vazia);

   (b) devolver o primeiro elemento de uma fila sem alterá-la;

   (c) verificar se uma fila está vazia;

   (d) inserir um elemento em uma fila;

   (e) remover um elemento de uma fila;

   (f) destruir uma fila.

6. Refaça o exercício anterior agora com pilha.

7. (Bom!) Suponha que você tenha disponível na memória do seu computador um vetor v com $N = 1000$ (ou outro número que você goste mais) posições livres e que você tenha que implementar simultaneamente duas pilhas nesse espaço livre. Mostre como fazer isso de modo a não desperdiçar espaço livre, isto é, se for feito um pedido de inserção em alguma das pilhas e o espaço ocupado pelas duas pilhas for menor que 1000, a inserção deve ser feita. Você deve descrever procedimentos/funções (para cada uma das pilhas) para inserir um elemento, remover um elemento e testar se a pilha está vazia.

8. Escreva uma função recursiva que lê uma expressão em notação pré-fixa e calcula a correspondente notação pós-fixa.

9. Uma sequência s de parênteses e colchetes é *balanceada* se:

   - s é vazia,
   - $s = (p)q$ onde p e q são sequências balanceadas, ou
   - $s = [p]q$ onde p e q são sequências balanceadas

   Escreva uma função recursiva que recebe uma sequência formada de parênteses e colchetes e verifica se ela é balanceada (devolve 0 ou 1).

10. A gramática das expressões em notação préfixa com operandos sendo variáveis $a, b, c, \ldots, z$ e operadores binários $+$, $-$, $*$, $/$ e $\wedge$ e operador unário $-$ (troca o sinal) pode ser definida recursivamente da seguinte forma. Uma expressão s é préfixa (i.e., está em notação préfixa) se:

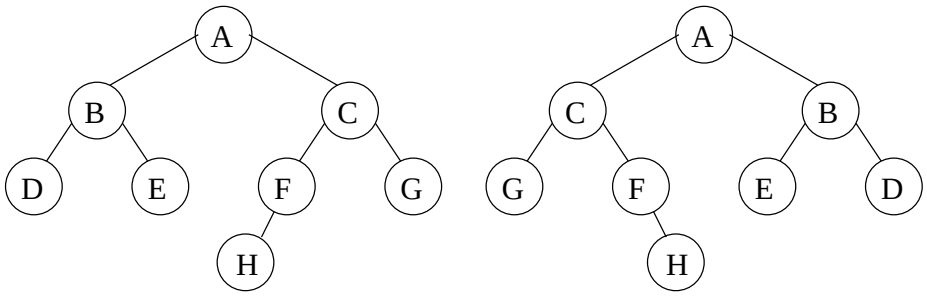
    - $s = x$ onde x é uma variável,
    - $s = \tilde{}p$ onde $\tilde{}$ representa o operador unário $-$ e p é uma expressão préfixa,
    - $s = \oplus pq$ onde $\oplus$ é $+$ ou $-$, e p, q são expressões préfixas,
    - $s = \otimes pq$ onde $\otimes$ é $*$ ou $/$, e p, q são expressões préfixas, ou
    - $s = \wedge pq$ onde p, q são expressões préfixas.

    Suponha seja dado um vetor `valor[]` com os valores das variáveis: `valor[0]` é o valor (inteiro) da variável a, `valor[1]` é o valor (inteiro) da variável b, etc. Escreva uma função que calcula o valor da expressão.

11. O *espelho* T' de uma árvore binária T é definida da seguinte forma:

    - se T é vazia, seu espelho é vazio;
    - senão T' tem a seguinte forma: a raiz de T' é a igual à raiz de T, a subárvore esquerda de T' é o espelho da subárvore direita de T e a subárvore direita de T' é o espelho da subárvore esquerda de T.

    A figura abaixo ilustra esta ideia:



    Escreva uma função recursiva `p_no Gira(p_no rais)` que recebe o apontador para uma árvore binária e a transforma em seu espelho. Nenhum nó deve ser criado e a função deve apenas alterar apontadores esq e dir.

12. Suponha que seja dada uma árvore de expressão aritmética. Os campos chave são do tipo char sendo ou variáveis $a, b, \ldots, z$ ou operadores binários $+, -, *, /$ e $\wedge$. Suponha ainda que você tem um vetor `int valor[]` com os valores das variáveis. Escreva uma função que avalia a expressão representada pela árvore. Use um dos percursos em profundidade (qual?).

13. Escreva uma versão da função de percurso em largura visto em aula que não enfileire NULL na fila.

14. Escreva os algoritmos de percurso com pilha explícita para pós-ordem e in-ordem similares ao visto em aula para pré-ordem.

15. Se você não fez assim no exercício anterior, refaça de modo que não empilhe NULL na pilha.

16. Como são as árvores binárias para as quais os percursos em pré-ordem e in-ordem produzem a mesma sequência?
17. Como são as árvores binárias para as quais os percursos em pós-ordem e in-ordem produzem a mesma sequência?
18. Em geral dada apenas o percurso em pré-ordem de uma árvore ninária, não é possível reconstruir a árvore original (de modo único). Por exemplo, há duas árvores binárias cujo percurso em pré-ordem é AB. Pode-se construir exemplos maiores onde isto acontece (tente!). Entretanto, se forem dados os percursos em pré-ordem e in-ordem, é possível reconstruir a árvore. Por exemplo, suponha que os percursos em pré-ordem e in-ordem sejam ABDECFHG e DBEAHFCG, respectivamente. Existe exatamente uma árvore com esses percursos, a árvore à esquerda na questão 10. Escreva uma função recursiva

    `p_no Reconstroi(char pos[ ], int ip, int fp, char in[ ], int ii, int fi)`

    que recebe os percursos em pós-ordem (`pos[ip..fp]`) e em inordem (`in[ii..fi]`) de uma árvore binária (não conhecida) e reconstrói a árvore (você deve devolver um apontador para a árvore reconstruída). Suponha que os nós são descritos por letras $A, B, \ldots, Z$. A chamada inicial é
    `p = Reconstroi(pos, 0, n-1, in, 0, n-1)`. A função não pode usar vetores auxiliares ou alocação adicional de memória proporcional ao número de nós.