Listas ligadas I

Nesta atividade você deverá criar algumas funções para manipulação de listas ligadas. No arquivo `listas.c` que você verá na sequência, existem algumas funções já implementadas: são aquelas que vimos durante a aula teórica. Outras funções estão marcadas como exercício: são as que você deverá implementar durante esta aula de laboratório.

Considere que as listas são simplesmente ligadas e que você só tem acesso aos parâmetros passados em cada função (não suponha que existam variáveis globais que guardam o inicio da lista nem o final). Ou seja, sua função deve trabalhar unicamente com o que foi passado a ela como parâmetro.

Arquivo lista.h

```
#ifndef LISTA
#define LISTA

typedef struct s_no *   no;

struct s_no {
  int item;
  no  prox;
};

  no novo             (int item);
void deleta           (no x);
void insere_inicio    (no *inicio,    no x);
void remove_inicio    (no *inicio);
void imprime          (no inicio);
  no busca            (no inicio,     int item);
  no buscaR           (no inicio,     int item);
  no final            (no inicio);
void insere_final     (no *inicio,    no x);
void insere_finalR    (no *inicio,    no x);
void remove_um        (no *inicio,    int item);
void remove_todos     (no *inicio,    int item);
void remove_todosR    (no *inicio,    int item);
  no copia            (no inicio);
void inverte          (no *inicio);
void inverteR         (no *head,      no *tail);
void inverteR2        (no *head);

no livres = NULL;

#endif
```

Arquivo lista.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

// 1. (EXERCÍCIO) Cria um novo nó da lista.
//    Mudar a implementação da função novo
//    para alocar muitos nós de uma vez e usar a lista
//    livres para guardar os blocos que ainda não estão
//    sendo usados. Como visto em sala...
no novo(int item);

// 2. (EXERCÍCIO) Deleta um nó.
//    Adaptar a função deleta para trabalhar
//    com a lista de nós livres. Como visto em sala...
void deleta(no x);

// 3. Insere um nó no início da lista (como PUSH).
//    (Supõem que x e ini são ambos diferentes de NULL.)
//    (Mas *ini pode ser NULL.)
void insere_inicio(no *ini, no x) {
  x->prox = *ini;
  *ini = x;
}

// 4. Remove o primeiro nó da lista.
//    (Se o valor de retorno fosse o item do nó removido,
//    essa função seria como POP.)
void remove_inicio(no *ini) {
  no x = *ini;
  if (*ini != NULL) {
    *ini = x->prox;
    deleta(x);
  }
}

// 5. Imprime todos os elementos de uma lista
//    (supondo que sejam inteiros).
void imprime(no x) {
  for (; x != NULL; x = x->prox)
    printf("%d", x->item);
  printf("\n");
}

// 6. Busca o primeiro nó contendo item.
//    Retorna NULL se não encontrar o nó.
no busca(no inicio, int item) {
```

```
  for (; inicio != NULL && inicio->item != item;
       inicio = inicio->prox);
  return inicio;
}

// 7. Busca o primeiro nó contendo item, recursivamente.
//    Retorna NULL se não encontrar o nó.
no buscaR(no inicio, int item) {
  if (inicio == NULL)
    return NULL;
  if (inicio->item == item)
    return inicio;
  return buscaR(inicio->prox, item);
}

// 8. Devolve o último nó de uma lista.
no final(no x) {
  if (x == NULL)
    return NULL;

  while (x->prox != NULL)
    x = x->prox;

  return x;
};

// 9. Insere nó no final.
//    (Supõem que x e ini são ambos diferentes de NULL.)
//    (Mas *ini pode ser NULL.)
void insere_final(no *ini, no x) {
  if (*ini == NULL) {
    *ini = x;
    return;
  }

  no y = final(*ini);
  y->prox = x;
}

// 10. Insere nó no final, recursivamente.
//    (Supõem que x e ini são ambos diferentes de NULL.)
//    (Mas *ini pode ser NULL.)
void insere_finalR(no *ini, no x) {
  if (*ini == NULL)
    *ini = x;
  else
    insere_finalR(&((*ini)->prox), x);
}
```

```c
// 11. Remove primeiro nó que contém item.
//     (Supõem que ini é diferente de NULL.
//     (*ini pode ser NULL.)
//     (EXERCÍCIO: entender o que a função faz.)
void remove_um(no *ini, int item) {
  if (*ini == NULL)
    return;

  no x, *prev = ini;
  for (x = (*ini); x != NULL && x->item != item;
       prev = &(x->prox), x = x->prox);

  if (x != NULL) {
    *prev = x->prox;
    deleta(x);
  }
}

// 12. (EXERCÍCIO) Remove todos os nós contendo item.
void remove_todos(no *ini, int item);

// 13. (EXERCÍCIO) Remove todos os nós contendo item, recursivo.
//     Este fica mais simples que o anterior.
void remove_todosR(no *ini, int item);

// 14. Cria uma cópia da lista dada
//     (copiar em outras posições de memória, é claro).
no copia(no inicio) {
  if (inicio == NULL)
    return NULL;

  no x = novo(inicio->item);
  x->prox = copia(inicio->prox);
  return x;
}

// 15. Inverte a lista.
void inverte(no *ini) {
  no x = *ini;
  *ini = NULL;

  while (x != NULL) {
    no y = x->prox;
    x->prox = *ini;
    *ini = x;
    x = y;
  }
```

```
}

// 16. (EXERCÍCIO) Função recursiva para inverter uma lista
//     Agora só com o ponteiro para (o ponteiro para) o
//     primeiro nó sendo passado como parâmetro.
void inverteR2(no *ini);
```