Listas ligadas

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.

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

typedef struct s_no *   no;

struct s_no {
  int item;
  no  prox;
};

// 1. Cria um novo nó da lista.
no novo(int item) {
  no x = (no) malloc(sizeof(struct s_no));

  if (x == NULL) {
    fprintf(stderr, "Memória insuficiente.\n");
    exit(1);
  }

  x->item = item;
  x->prox = NULL;

  return x;
}

// 2. Deleta um nó.
void deleta(no x) {
  free(x);
}
```

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

// 5. Imprime todos os elementos de uma lista
//    (supondo que sejam inteiros).
void imprime(no inicio) {
  no x;
  for (no x = inicio; 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 (EXERCÍCIO) Devolve o último nó de uma lista.
no final(no inicio);

// 9. (EXERCÍCIO) Insere nó no final.
//    (Supõem que x e inicio são ambos diferentes de NULL.)
//    (Mas *inicio pode ser NULL.)
void insere_final(no *inicio, no x);

// 10. (EXERCÍCIO) Insere nó no final, recursivamente.
//    (Supõem que x e inicio são ambos diferentes de NULL.)
//    (Mas *inicio pode ser NULL.)
void insere_finalR(no *inicio, no x);

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

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

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

// 11. Remove primeiro nó que contém item.
//     (Provavelmente a 1ª idéia que vem à mente.)
void remove_um_v1(no *inicio, int item) {
  if (*inicio == NULL)
```

```
    return;

  no x = *inicio, y;

  if (x->item == item) {
    *inicio = x->prox;
    deleta(x);
    return;
  }

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

  if (x->prox == NULL)
    return;

  y = x->prox;
  x->prox = y->prox;
  deleta(y);
}

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

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

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

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

  while (*inicio != NULL) {
    x = *inicio;
    *inicio = (*inicio)->prox;
    x->prox = inv;
    inv = x;
```

```
  }

  *inicio = x;
}

// 16. Função recursiva para inverter uma lista.
//     (EXERCÍCIO: entender o que a função faz.)
void inverteR(no *head, no *tail) {
  if (*head == *tail) return;
  inverteR(&((*head)->prox), tail);
  (*tail)->prox = *head;
  *tail = *head;
  *head = (*head)->prox;
  (*tail)->prox = NULL;
}

// 17. (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 *head);

/*****************************************************/
/*****************************************************/
/*****************************************************/

// Testa inverteR
int testa_inverteR() {
  no head = NULL, tail;

  for (int i = 0; i < 10; i ++) {
    insere_inicio(&head, novo(i));
    if (i == 0)
      tail = head;
  }

  imprime(head);
  inverteR(&head, &tail);
  imprime(head);

  return 0;
}
```

```c
// Troque o corpo da função main para testar outras funções...
int main() {
  return testa_inverteR();
}
```