```
/*
 * Faça uma biblioteca para as definições
 * abaixo especificadas.
 */

typedef int TipoChave;
typedef int TipoValor;
struct TipoListaSimples
{
  TipoChave chave;
  TipoValor valorQualquer;
  struct TipoListaSimples *prox;
};
typedef struct TipoListaSimples TipoListaSimples;

/*==================================>PROCEDIMENTOS BÁSICOS DE LISTAS
/* ------------------------> Insercao no início
 * Insere novo nó no início de uma lista encadeada.
 * A referência de ponteiro para o primeiro nó e os
 * valores dos campos do nó são dados.
 * Devolve endereço do nó recém inserido
 * ou NULL em caso de insucesso.
 */
TipoListaSimples *insereInicioListaSimples(TipoListaSimples **prim,
TipoChave chave, TipoValor valor);

/* ------------------------> Atualiza primeiro no
 * Caso o primeiro nó exista, atualiza o campo
 * valorQualquer com novoValor. É dado o ponteiro para
 * o primeiro nó.
 */
void atualizaValor(TipoListaSimples *prim, TipoValor novoValor);

/* ------------------------> Remoção no início
 * Remove o primeiro nó de uma lista
 * caso ele exista. O segundo, se existir, passará
 * a ser o primeiro.
 */
void removePrimeiroNo(TipoListaSimples **prim);

/* -----------------> Busca pelo endereço de um nó dado um valor de chave
 * Devolve ponteiro para o nó cujo valor chave é 'chave'
 * ou NULL caso este não exista.
 */
TipoListaSimples *pesquisaNo(TipoListaSimples *prim, TipoChave chave);


/* ------------------------> Inserção no fim de uma lista
 * Insere nó no fim de uma lista dada a referência do ponteiro
 * do primeiro nó.
 * Devolve endereço do novo nó ou NULL em caso de
 * insucesso.
 */
TipoListaSimples * insereFimListaSimples(TipoListaSimples **prim,
TipoChave chave);
```

```
/* -------------------------> Remoção último nó
 * Remove o último nó de uma lista  (caso ele exista) dada a
 * referência do ponteiro do primeiro nó.
 */
void removeUltimoNo(TipoListaSimples **prim);


/* -------------------------> Remove nó por valor de chave
 * Remove nó cujo valor chave seja igual a 'chave'
 * Mantêm lista inalterada caso este não exista.
 */
void removeNo(TipoListaSimples **prim, TipoChave chave);

/* -------------------------> Remove todos nós
 * Remove TODOS os nos da lista exceto e atualiza
 * ponteiro para o primeiro para NULL.
 * Dica: recursividade pode ajudar muito!
 */
void liberaNos(TipoListaSimples **prim);

/*================================>PROCEDIMENTOS ESPECÍFICOS DE LISTAS
/* -------------------------> Cria cópia
 * Cria uma nova lista cujos nós têm os mesmos
 * valores da lista dada. Devolve o ponteiro para
 * o primeiro nó da nova lista.
 */
TipoListaSimples *copiaListas(TipoListaSimples *prim);

/* -------------------------> Cria cópia
 * Calcula a interseção entre as duas listas
 * dadas e insere tais nós numa (nova) terceira
 * lista. Devolve o ponteiro para o cabeça da nova lista.
 * A interseção deve considerar o campo chave.
 */
TipoListaSimples *intersecaoListas(TipoListaSimples *prim1,
TipoListaSimples *prim2);

/* -------------------------> Remove/Insere
 * Remove último nó da segunda lista (caso ele exista).
 * Insere um novo nó na última posição da primeira lista.
 * O novo nó deve ter os mesmos valores do nó removido.
 * A referencia do ponteiro para o primeiro nó das listas de entrada
 * são primLista1 eprimLista2, respectivamente.
 */
void insereRemove(TipoListaSimples **primLista1, TipoListaSimples
**primLista2);

/* -------------------------> Transplanta Nó.
 * Desconecta o último nó da segunda lista (caso ele exista).
 * Conecta tal nó na última posição da primeira lista.
 * NOTE QUE A REGIÃO DE MEMÓRIA DO NÓ NÃO MUDA!
 * A referencia do ponteiro para o primeiro nó das listas de entrada
 * são primLista1 eprimLista2, respectivamente.
 */
void transplantaNo(TipoListaSimples **primLista1, TipoListaSimples
**primLista2);
```

```
/* -------------------------> Conta Nó.
 * Conta o número de nós de uma lista encadeada.
 * Retorno número de nós
*/

int contaNo(TipoListaSimples **primLista1);

/* -------------------------> Altura do Nó.
* Escreva uma função que calcule a altura de um dado nó (baseado na
*chave).
* A altura de um nó 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 último nó da lista.
* Retorno altura do nó
*/

int alturaNo(TipoListaSimples **primLista1, TipoChave chave);

/* -------------------------> Profundidade do Nó.
* Escreva uma função que calcule a profundidade de um dado nó (baseado na
*chave).
*A profundidade de um nó c em uma lista encadeada é o número de passos do
*único caminho que vai do primeiro nó da lista até c.

int profundidadeNo(TipoListaSimples **primLista1, TipoChave chave);
```