Programação Estruturada – lista 2 de exercícios

Listas Encadeadas

Em todos os exercícios a seguir, teste para os casos: a) lista vazia; b) com apenas um elemento; c) caso geral

0. No arquivo `noint.c`, implemente

```
int noint_comprimento(struct NoInt *cabeca);
```

que retorna o número de nós na lista cujo nó cabeça é dado.

1. No arquivo `noint.c`, implemente

```
struct NoInt *noint_iesimo(struct NoInt *cabeca, int i);
```

que retorna o primeiro nó da lista de `i` for 0, o segundo se `i` for 1, o terceiro se `i` for 2, e assim em diante. Se não existir tal nó na lista, retorna `NULL` .

2. No arquivo `noint.c`, implemente

```
struct NoInt *noint_busca(struct NoInt *cabeca, int dado);
```

que retorna o primeiro nó encontrado que possui o dado fornecido ou `NULL` se não há nó com aquele dado na lista.

3. No arquivo `noint.c`, implemente

```
struct NoInt *noint_remove_no(struct NoInt *cabeca,
                              struct NoInt *aremover);
```

se o nó passado no ponteiro `aremover` estiver na lista, remove-o da lista e retorna o ponteiro para o novo nó cabeça (ou para o mesmo nó cabeça, se este não foi removido). Lembre-se de ajustar os ponteiros dos nós adjacentes e liberar memória.

Se o nó `aremover` não for um nó da lista, retorna apenas o nó cabeça sem fazer nada. O mesmo deve ser feito se `aremover` for `NULL` .

4. No arquivo `noint.c`, implemente

```
struct NoInt *noint_remove(struct NoInt *cabeca, int dado);
```

que remove o primeiro nó encontrado com o dado fornecido e retorna o ponteiro para o nó cabeça atualizado (pode mudar se nó cabeça for removido).

5. No arquivo `noint.c`, implemente

```
struct NoInt *noint_remove_todos(struct NoInt *cabeca, int dado);
```

que remove todos os nós encontrados com o dado fornecido e retorna o ponteiro para o nó cabeça atualizado.

6. No arquivo `noint.c`, implemente

```
struct NoInt *noint_insere_ordenado(struct NoInt *cabeca, int n);
```

que insere nó com número `n` logo após o último nó cujo dado é menor ou igual a `n`.

7. No arquivo `noint.c`, implemente

```
struct NoInt *noint_ordena(struct NoInt *cabeca);
```

que retorna uma nova lista construída com os dados da lista fornecida, mas de forma ordenada.

8. No arquivo `noint.c`, implemente

```
struct NoInt * noint_bubblesort(struct NoInt *cabeca);
```

que ordena a lista, usando o bubblesort e retorna o ponteiro para o novo nó cabeça. A ordenação deve ser feita ***in-place***, ou seja não é permitido criar outra lista, nem criar outros nós.

9. Um inconveniente de se representar uma lista ligada pelo seu nó cabeça é que a maioria das operações pode, potencialmente, mudar o ponteiro para o nó cabeça, o que faz com que tenhamos que retornar um ponteiro o novo nó cabeça e atualizar manualmente a variável que mantém o endereço do nó cabeça.

Para evitar esse inconveniente, crie um arquivo `listaint.c` e defina nele uma struct:

```
#include <noint.c>

struct ListaInt {
  struct NoInt *cabeca;
};

void listaint_constroi(struct ListaInt *lista) {
  lista->cabeca = NULL;
}
```

Implemente neste arquivo as funções a seguir:

```
/**
 * Desaloca todos os nós da lista e deixa-a vazia.
 */
void listaint_destroi(struct ListaInt *lista);

/**
 * Insere um elemento no início da lista. Retorna verdadeiro
 * se a inserção for bem sucedida, falso caso contrário.
 */
int listaint_insere_inicio(struct ListaInt *lista, int dado);

/**
 * Remove o primeiro nó da lista. Retorna verdadeiro
 * se a remoção for bem sucedida, falso caso contrário.
 */
int listaint_remove_inicio(struct ListaInt *lista, int dado);
```

```c
/**
 * Retorna o último nó da lista ou NULL se ela estiver vazia.
 */
struct NoInt *listaint_ultimo_no(struct ListaInt *lista, int dado);

/**
 * Insere um elemento no final da lista. Retorna verdadeiro
 * se a inserção for bem sucedida, falso caso contrário.
 */
int listaint_insere_fim(struct ListaInt *lista, int dado);

/**
 * Remove um elemento no final da lista. Retorna verdadeiro
 * se a inserção for bem sucedida, falso caso contrário.
 */
int listaint_remove_fim(struct ListaInt *lista);

/**
 * Retorna o número de nós na lista.
 */
int listaint_comprimento(struct ListaInt *lista);

/**
 * Retorna o i-ésimo nó da lista, onde i=0 retorna o primeiro.
 * Se o nó não existir na lista, retorna NULL.
 */
struct NoInt *listaint_iesimo(struct ListaInt *lista, int i);

/**
 * Retorna o primeiro nó da lista que possui o dado fornecido.
 * Se não existir nó na lista com tal dado, retorna NULL.
 */
struct NoInt *listaint_busca(struct ListaInt *lista, int dado);

/**
 * Remove o nó fornecido. Retorna verdadeiro se o nó foi
 * removido, falso se o nó fornecido não é um nó da lista.
 */
int listaint_remove_no(struct ListaInt *lista, struct NoInt *aremover);

/**
 * Remove o nó com dado fornecido. Retorna verdadeiro se nó foi
 * removido, falso se o dado fornecido não está na lista.
 */
int listaint_remove(struct ListaInt *lista, int dado);

/**
 * Remove todos os nós que possuem  dado fornecido. Retorna
 * verdadeiro se nó foi removido, falso se o dado fornecido não
 * está na lista.
 */
int listaint_remove_todos(struct ListaInt *lista, int dado);
```

```
/**
 * Insere nó com valor n após o último nó cujo dado é menor
 * ou igual a n. Retorna verdadeiro se a operação foi bem
 * sucedida, falso caso contrário.
int listaint_insere_ordenado(struct ListaInt *lista, int n);
```

10. Um inconveniente de se manter apenas uma referência ao nó cabeça de uma lista ligada é que as operações realizadas no fim da lista requerem maior tempo de execução, uma vez que é necessário percorrer toda a lista para encontrarmos o último nó.

Para evitar esse inconveniente, é possível criar uma estrutura de dados chamada *deque* (*Double–Ended QUEue*, ou seja, fila com dois extremos).

Um deque para inteiros pode ser definido da seguinte forma:

```
#include <noint.c>

struct DequeInt {
  struct NoInt *cabeca;
  struct NoInt *cauda;
};

void dequeint_constroi(struct ListaInt *lista) {
  lista->cabeca = NULL;
  lista->cauda = NULL;
}
```

Reimplemente as funções `listaint...` para deque, de tal forma que as operações realizadas no final da lista (ultimo_no, inserção, remoção) sejam feitas sem que a lista inteira seja percorrida.