Universidade do Estado de Santa Catarina (UDESC)
Curso: Tecnologia em Análise e Desenvolvimento de Sistemas
Disciplina: Estruturas de Dados (EDA0001)
Prof. Rui J. Tramontin Jr.

Lista de Exercícios 5 (Listas Encadeadas)

1) Faça uma função que retorna a quantidade de elementos em uma lista. Cabeçalho:

```
int conta_elementos(Lista l)
```

2) Faça uma função para retornar o índice da maior informação contida em uma lista. Cabeçalho:

```
int maior_valor(Lista l, int (*compara)(void*, void*) )
```

3) Faça uma função para esvaziar uma lista, ou seja, que desaloca todos os elementos e suas respectivas informações. Cabeçalho:

```
void esvazia_lista(Lista *l)
```

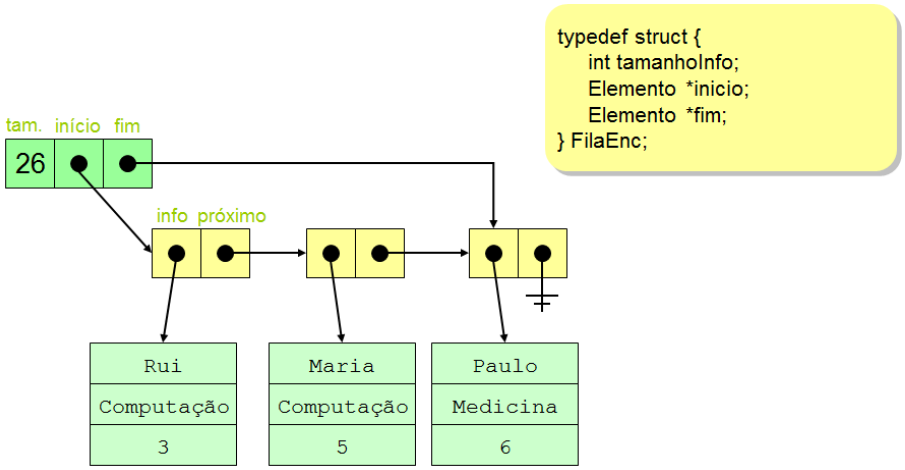
4) Implemente uma função que mostre uma **lista duplamente encadeada** na ordem inversa.

5) Escreva uma função que mostre uma **lista encadeada simples** na ordem inversa. Faça diferentes versões:
    a) Usando uma estrutura auxiliar, tal como uma pilha;
    b) Usando recursão (a pilha é usada implicitamente);
    c) Usando estrutura de repetição (sem pilha).

6) Faça uma função que retorna a posição de um elemento. Será preciso percorrer a lista até que se encontre o elemento cuja informação seja igual à informação passada como parâmetro. Retorna -1 caso a lista não contenha a informação. Cabeçalho:

```
int posicaoDoElemento(Lista *l, void *info, int (*comparaInfo)(void *, void *))
```

7) Implemente um TAD que representa uma **Fila** usando uma **lista encadeada**. A estrutura do cabeçalho da lista deve ser modificada para fazer referência para o último elemento. Além disso, é preciso modificar as funções de manipulação para levar em conta tal referência. A função "inserir" é implementada por "adicionaNoFim" e a função "remover" implementada por "removeDoInicio". A estrutura pode ser **simples** ou **duplamente encadeada**. Como exemplo, temos uma fila construída com base numa *lista simplesmente encadeada*, conforme segue:



8) Implemente uma **fila de prioridades** com base numa lista encadeada, implementando as três alternativas diferentes:

   a. *Lista simplesmente encadeada*: função *remover* permanece a mesma (no início); função *inserir* é implementada pela função "insere em ordem".

   b. *Lista simplesmente encadeada*: função *inserir* permanece a mesma (no fim); função *remover* deve buscar o menor valor da lista e removê-lo.

   c. *Lista duplamente encadeada*: função *remover* permanece a mesma (no início); função *inserir* é implementada pela função "insere em ordem", mas o percurso para a inserção é feito a partir do último elemento.

9) Escreva uma função que **inverte** uma **lista duplamente encadeada**. A função deve proceder da seguinte maneira:

   a) Crie dois ponteiros: **p1**, apontando para o primeiro elemento, e **p2**, para o último;

   b) Usando um ponteiro auxiliar, troque as respectivas informações dos dois elementos;

   c) Faça **p1** apontar para o seu sucessor, bem como **p2** apontar para o seu antecessor;

   d) Repita os passos *b* e *c* até **p1** e **p2** "se cruzarem".

10) Implemente uma função que **ordena** uma **lista encadeada simples**. Este exercício toma como base o algoritmo ***SelectionSort***, implementado para um vetor. O exercício consiste em adaptar a noção de vetor (e índices) para a noção de elementos (e ponteiros para o próximo elemento). Em outras palavras, ao invés de guardarmos o índice de um dado elemento do vetor, guardamos o ponteiro que aponta para o elemento da lista encadeada.

```
void selectionSort(int v[], int tamanho){
   int i;
   for(i = 0; i < tamanho - 1; i++){
      int i_menor = i;
      int j;
      for(j = i + 1; j < tamanho; j++){
         if(v[j] < v[i_menor]){
            i_menor = j;
         }
      }
      int temp = v[i];
      v[i] = v[i_menor];
      v[i_menor] = temp;
   }
}
```