**Universidade Federal de Ouro Preto – UFOP**
**Instituto de Ciências Exatas e Biológicas – ICEB**
**Departamento de Computação – DECOM**
**Disciplina: Algoritmos e Estruturas de Dados I – CIC102**
**Professor: David Menotti (menottid@gmail.com)**

# Lista de Exercícios sobre Listas Implementadas por Encadeamento

1) Se você tem de escolher entre uma representação por lista encadeada ou uma representação usando posições contíguas de memória para um vetor, quais informações são necessárias para você selecionar uma reprentação apropriada? Como esses fatores influenciam a escolha da representação?

2) Considere listas implementadas por encadeamento simples, então pede-se para implementar funções que:
   a. Localize/ Pesquise/Encontre (*search*) elementos
   b. Concatenar/intercalar (*Merge*) duas listas
   c. Dividir uma lista em várias ($k$)
   d. Copiar uma lista
   e. Ordenar (*sort*) ima lista por ordem crescente/decrescente

3) Considere listas implementadas por encadeamento duplo, entao pede-se para implementar funções que:
   a. Localize/ Pesquise/Encontre (*search*) elementos
   b. Concatenar/intercalar (*Merge*) duas listas
   c. Dividir uma lista em várias ($k$)
   d. Copiar uma lista
   e. Ordenar (*sort*) ima lista por ordem crescente/decrescente

4) Escreva uma função em C para trocar os elementos $m$ e $n$ de uma lista simplesmente encadeada ($m$ e $n$ podem ser chaves ou mesmo ponteiros para os elementos – a escolha é sua).

5) Agora escreva uma função em C para trocar os elementos $m$ e $n$ de uma lista duplamente encadeada ($m$ e $n$ podem ser chaves ou mesmo ponteiros para os elementos – a escolha é sua).

6) Escreva uma rotina, *inssub*($l1$,$i1$,$l2$,$i2$,*len*) para inserir os elementos da lista $l2$, começando no elemento $i2$ e continuando por *len* elementos na lista $l1$, começando na posição $i1$. Nenhum elemento da lista $l1$ deverá ser removido ou substituído. Se $i1 > length(l1) + 1$ (onde $length(l1)$ indica o número de nós na lista), ou se $i2 + len - 1 > length(l2)$, ou se $i1 < 1$, ou se $i2 < 1$, imprima uma mensagem de erro. A lista $l2$ deve permanecer inalterada.

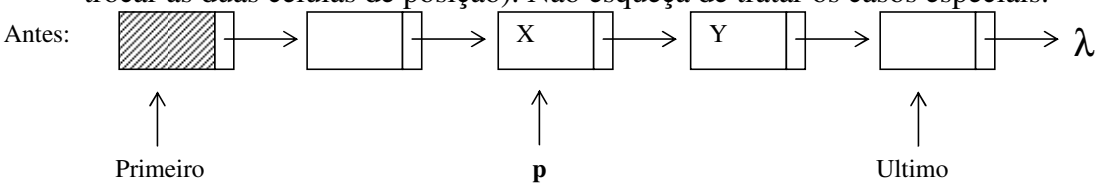
7) Escreva uma função em C, *search*($l$,$x$), que receba um ponteiro $l$ para uma lista de inteiros e um inteiro $x$, e retorne um ponteiro para um nó contendo x, se existir, e o ponteiro nulo, caso contrário. Escreva outra função, *searchinsert*(l,x), que inclua $x$ em $l$ se ele não for encontrado, e retorne sempre um ponteiro para um nó contendo $x$.

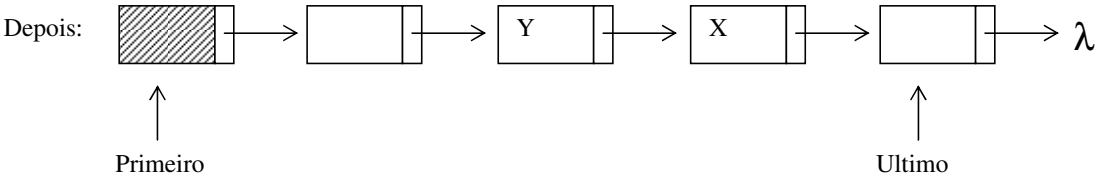
8) Suponha que uma string de caracteres seja representada por uma lista de caracteres individuais. Escreva um conjunto de funções para manipular estas listas como segue

(*l*1, *l*2 e *list* são ponteiros para um nó de cabeçalho de uma lista representando uma *string* de caracteres, *str* é um vetor de caracteres e *i*1 e *i*2 são inteiros):

a. *strcnvcl*(*str*) para converter a *string* de caracteres, *str*, numa lista. Essa função retorna um ponteiro para um nó de cabeçalho.
b. *strcnvlc*(*list*,*str*) para converter uma lista em uma *string* de caracteres.
c. *Strcmpl*(*l1*,*l2*) para comparar duas strings de caracteres, representadas or listas. Essa função retorna -1 se a *string* de caracteres representada por *l*1 é menor que a *string* representada por *l*2; 0 se são iguais, e 1 se a *string* representada por *l*1 é maior.

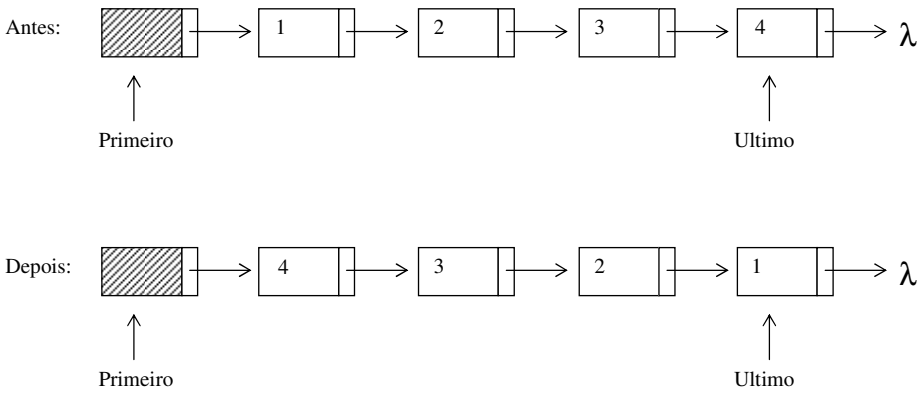
9) Considere a implementação de listas encadeadas utilizando apontadores vista em sala. Escreva um procedimento `Troca(TipoLista* pLista, TipoCelula* p)` que, dado um apontador para uma célula qualquer (**p**), troca de posição essa célula com a sua célula seguinte da lista, como mostrado na figura abaixo. (Obs. Não vale trocar apenas o campo item! Você deverá fazer a manipulação dos apontadores para trocar as duas células de posição). Não esqueça de tratar os casos especiais.



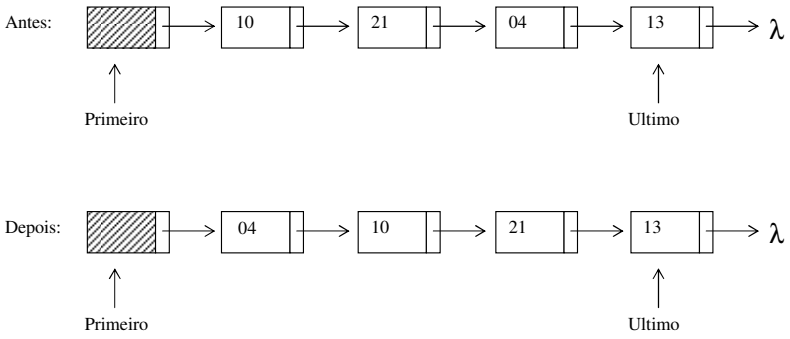




10) Considere a implementação de listas encadeadas utilizando apontadores vista em sala e responda as seguintes perguntas:

a. Escreva uma função `void Inverte(TipoLista *Lista)` que, dada uma lista com um número qualquer de elementos, inverte a ordem dos elementos da lista, como exemplificado na figura abaixo. (Obs. Não vale trocar apenas os campos item ou usar uma lista / fila / pilha auxiliar! Você deverá fazer a manipulação dos apontadores para trocar as células de posição).
b. Qual é a ordem de complexidade da sua função. Explique.




11) Escreva uma função `void MoveMenor(TipoLista Lista)` que, dada uma lista com um número qualquer de elementos, acha o menor elemento da lista e o move para o começo da lista, como exemplificado na figura abaixo. (Obs. Não vale trocar apenas os campos item ou usar uma lista / fila / pilha auxiliar! Você deverá fazer a manipulação dos apontadores para trocar as células de posição). Qual é a ordem de complexidade da sua função. Explique.

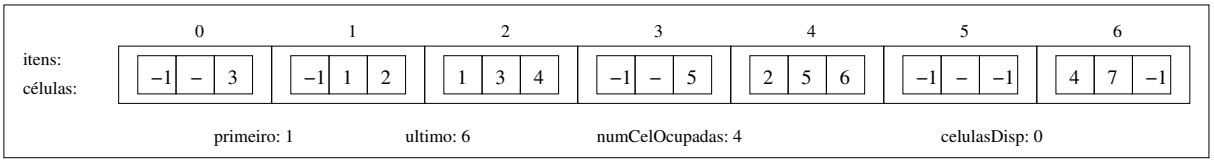



12) (Extraído e modificado de [3] apud [2]) Implemente o tipo abstrato de dados *Area*, cuja finalidade é gerenciar uma área interna de memória de forma que o maior e o menor elemento possam ser **removidos** dessa área a um custo O(1). A estrutura de dados que deve ser utilizada é uma lista linear duplamente encadeada implementada por **cursores**. Os cursores são números inteiros que representam posições em um arranjo e são utilizados para simular os apontadores da implementação tradicional das listas lineares duplamente encadeadas. A utilização de cursores evita a alocação e a liberação dinâmica de itens de memória, sendo mais eficiente em aplicações muito dinâmicas em que o número máximo de itens é conhecido.
O tipo abstrato de dados *Area* possui as seguintes operações:

a. Criar uma área de memória interna vazia.
b. Obter o número de células ocupadas na área de memória
c. Inserir um item de dado na área interna de memória, mantendo os itens ordenados.
d. Retirar o primeiro item da área de memória.
e. Retirar o último item da área de memória
f. Imprimir o conteúdo da área de memória.

A retirada de um item de uma lista duplamente encadeada pode ser realizada a um custo constante, desde que se conheça previamente o endereço do item na lista. Ao

manter a lista ordenada, os elementos de menor e de maior chave estão na primeira e na última posição respectivamente. A figura abaixo ilustra uma lista com capacidade máxima de sete itens.




Após a realização de várias inserções e remoções, os itens contidos na lista apresentada na figura acima possuem as chaves 1, 3, 5 e 7. Os itens de dados da lista linear duplamente encadeada são armazenados em um vetor do tipo *Celula*. Cada entrada do vetor contém uma estrutura que armazena um item de dado, um cursor que aponta para a célula que sucede aquela entrada (*prox*) e um cursor que aponta para a célula que antecede aquela entrada (*ant*). Além disso, são representados dois cursores, *primeiro* e *ultimo*, que apontam para a primeira e para a última célula da lista, respectivamente. Para facilitar o controle de quando a lista se encontra cheia ou vazia, utilize os campo *numCelOcupadas*, que indica quantas células da lista estão ocupadas.

O código abaixo mostra a declaração do tipo abstrato de dados Area. As células armazenados na estrutura *Area* precisam ser mantidas ordenadas. Observe que você deverá implementar uma função que permita comparar dois itens, visto que os itens são inseridos e mantidos ordenados.

```
#define TAMCELS 7
typedef struct {
  int chave;
  /* outros campos */
} Item;
typedef struct {
  Item item;
  int prox, ant;
} Celula;
typedef struct {
  Celula itens[TAMCELS] ;
  int celulas Disp, primeiro, ultimo;
  int numCelOcupadas;
};
```

Somente o que foi apresentado até agora não é suficiente para implementar o tipo abstrato de dados *Area*. Isso porque, para incluir um novo item de dado na lista, é necessário haver células disponíveis a fim de que a inserção seja realizada. Assim, para gerenciar a lista de células disponíveis em determinado instante, basta incluir um cursor na representação da estrutura de dados *Area*, o qual irá apontar para a primeira célula disponível. Tal cursor foi denominado *celulasDisp*. Como a lista ilustrada pela figura acima possui capacidade para sete itens/células e *numCelOcupadas* = 4, então existem três células disponíveis. A primeira delas é apontada por *celulasDisp*, ou seja, o índice zero do vetor de itens, a segunda é indicada pelo cursor *prox* da célula apontada por *celulasDisp*, ou seja, o índice três do vetor de itens/células. A terceira e última célula disponível é apontada pelo cursor *prox* da segunda e se encontra no índice cinco do vetor de itens/células. Ela é a última, pois o seu cursor *prox* possui o valor -1, o que indica a falta de um sucessor para ela.

Dessa forma, antes de incluir um novo item de dado em *Area*, remove-se a primeira célula da lista de disponíveis (apontada por *celulasDisp*) e a insere

ordenamente na lista linear duplamente encadeada que armazena os itens de dados de *Area* (o custo desta inserção não é O(1), pode ser O(*n*) no pior caso). Já ao remover um item de dados de *Area*, a célula que continha tal item deve ser inserida na lista de disponíveis. Para que a inserção e a remoção da **lista de disponíveis** seja realizada a um custo constante, elas devem ser realizadas na posição apontada por *celulasDisp*.

## Exercícios extraídos de (Referências)

[1] Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein, *Estruturas de Dados Usando C*, Makron Books/Pearson Education, 1995.
[2] N. Ziviani, F.C. Botelho, Projeto de Algoritmos com implementações em Java e C++, Editora Thomson, 2006.
[3] F.C. Botelho, Comunicação Pessoal, Belo Horizonte, MG, Brazil, 2003.