**Lista de Exercícios I - ED II**

**Data Limite de Entrega – 05/04/2024**

**As listas de exercícios são para serem resolvidas em duplas. Cuidado** **com as cópias de trabalhos, pois cópias surtirão na divisão de uma nota** **pela quantidade de trabalhos iguais. As respostas aos problemas aqui** **tratados envolvem programação. Sendo assim, cada exercício é tratado** **como um programa diferente. Somente os arquivos-fonte devem ser** **enviados pelo SIGAA. Para o envio é necessário se compactar TODOS os** **arquivos fontes construídos e então enviar este arquivo compactado pelo** **SIGAA. Quando o exercício não solicitar explicitamente o tipo de lista** **encadeada a ser utilizado (simplesmente, duplamente e circular) ficará a** **cargo da dupla decidir qual utilizar.**

**Exercício 1:** Implemente um algoritmo que trabalhe como um DEQUE e controle a inserção e remoção de valores numéricos inteiros em uma lista encadeada. Um menu deve ser construído com as opções de inserção e remoção que o usuário pode escolher para uma das 2 extremidades da lista. Após a execução de uma das opções do menu o programa deve retornar para o menu principal para ser escolhida outra opção. Segue as funcionalidades do programa:

* Inserção no topo: esta inserção, como a da pilha, deve ser realizada na frente dos valores da lista encadeada, logo após o ponteiro que indica o início da lista encadeada;
* Inserção no final: assim como na fila, a inserção deve ser realizada na última posição

válida da lista encadeada;

* Remoção no topo: assim como a remoção em uma pilha e em uma fila, esta opção

deve remover o primeiro elemento da lista encadeada, que se encontra logo no início da lista encadeada;

* Remoção no final: esta opção, é diferente das estruturas de dados pilha e fila, esta opção deve remover o elemento que se encontra ao final da lista encadeada, como o último elemento da lista encadeada. Um detalhe, para esta remoção NÃO SE DEVE PERCORRER A LISTA ENCADEADA;
* Verificar valor inicial: assim como a pilha e a fila, o programa deve retornar ao usuário o valor que se situa no início da lista encadeada, ou seja, o primeiro elemento;
* Verificar valor final: diferentes das estruturas convencionais, o programa deverá retornar ao usuário o valor que se encontra no final da lista encadeada, ou seja, o último elemento. Um detalhe, para esta verificação NÃO SE DEVE PERCORRER

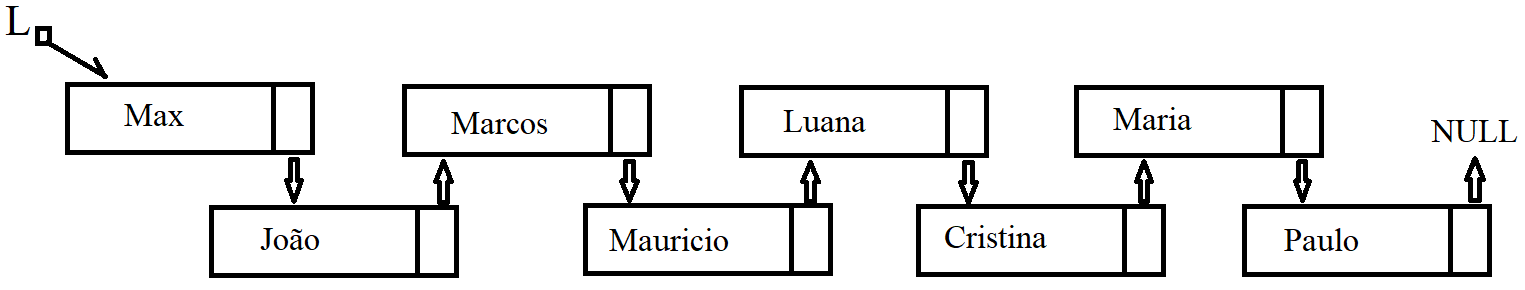
A LISTA ENCADEADA;

* Fim: o programa encerra sua execução.

**Exercício 2:** implemente uma lista encadeada que funcione como um registro de funcionários de uma empresa, mas que armazene indivíduos de 3 hierarquias diferentes: gerentes, administrativos e técnicos. Todos os elementos das 3 hierarquias devem estar trabalhando dentro de uma mesma lista. A lista NÃO PODE conter dados de elementos de diferentes hierarquias embaralhados. A lista pode ser construída com nomes. Aqui a ideia é controlar uma lista contendo conjuntos de dados de mesma natureza, mas divididos em 3 características diferentes, que neste caso é a hierarquia de trabalho. Um exemplo está a seguir:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Gerente | | Administrativos | | | Técnicos | |  |
| Max | João | Marcos | Mauricio | Luana | Cristina | Maria | Paulo |

Neste caso, diferente de um vetor estático, não temos mais as designações das posições pois a lista encadeada não funciona da mesma maneira. Graficamente, para uma lista simplesmente encadeada, a estrutura acima poderia se apresentar da seguinte forma:



Acontece que temos uma suborganização dentro desta lista, realizada pela hierarquia. Assim, a hierarquia é MAIS UM DADO a ser adicionado A CADA NÓ, além do nome e ponteiros da lista encadeada utilizada. Uma estratégia é ter uma única cabeça para a lista, como a imagem anterior indicou. Outra estratégia é se criar ALÉM do ponteiro que indica a cabeça da lista, que indica os elementos da primeira hierarquia diretamente, outros 2 ponteiros, que se situarão NO MEIO da lista encadeada, e que CADA UM destes ponteiros indicam o primeiro elemento das OUTRAS hierarquias. Assim, ou você pode percorrer a lista encadeada inteira utilizando o ponteiro da cabeça OU percorrer os elementos de CADA hierarquia separadamente.

As funcionalidades são as seguintes:

* Inserção: a função de inserção deve solicitar para QUAL hierarquia o

indivíduo a ser cadastrado pertence, e daí então solicitar o nome a ser inserido. A inserção, em cada hierarquia deve sempre ocorrer no final dos elementos DAQUELA hierarquia;

* Listar Todos os Servidores: o programa deve apresentar todos os dados   
  da lista POR HIERARQUIA, informando a qual hierarquia os servidores fazem parte, e seus nomes;
* Listar Servidores Por Hierarquia: o programa deve apresentar todos os

dados da lista encadeada (nomes), somente da hierarquia escolhida;

* Sair: o programa encerra sua execução.

**Exercícios 3:** dada uma lista encadeada L desordenada, de elementos inteiros distintos (todos diferentes entre si) inseridos de forma automática (utilizar a ferramenta rand() para números aleatórios), crie uma lista encadeada K ordenada, com os mesmos nós da lista L. A função remove os elementos da lista L, sempre do maior para o menor, e insere-os no início da lista K, que dessa maneira acaba se tornando uma lista ordenada de forma crescente. Não devem ser criados nós extras. Os nós utilizados em K devem ser OS MESMOS da lista

L. No final do processo a lista L estará vazia e a lista K conterá os nós anteriormente alocados para a lista L ordenados na forma crescente. Assim, crie a lista L, peça que o usuário insira a quantidade de elementos a serem inseridos de forma automática em L, apresente os elementos da lista encadeada L ao usuário, e depois apresente a lista encadeada K com os elementos de L ordenados.

**Exercício 4:** implemente uma lista SIMPLESMENTE encadeada de números inteiros. Deve ser solicitado ao usuário que insira a quantidade de elementos para esta lista, e os elementos devem ser inseridos de forma automática de maneira aleatória (função rand()), sem preocupação se existirão ou não elementos repetidos. Após construída a lista simplesmente encadeada e inseridos seus elementos, construa uma função que INVERTA a direção das ligações dos ponteiros. Ao final apresente os elementos desta lista com as ligações invertidas.

**Exercício 5:** o problema de Josephus descreve a seguinte situação onde um grupo de soldados está cercado e não há esperança de vitória, porém existe somente um cavalo disponível para escapar e buscar por reforços. Para determinar qual soldado deve escapar para encontrar ajuda, eles formam um círculo e sorteiam um número de um chapéu. Começando por um soldado sorteado aleatoriamente, uma contagem é realizada até o número sorteado. Quando a contagem terminar, o soldado em que a contagem parou é removido do círculo, um novo número é sorteado e a contagem recomeça no soldado seguinte ao que foi eliminado. A cada rodada, portanto, o círculo diminui em um, até que somente um soldado reste e seja escolhido para a tarefa.

Utilizando uma LISTA CIRCULAR, solicite ao usuário para inserir o tamanho desta lista, e a inserção dos elementos deve ser feita de forma automática. O tipo de item deve ser um valor numérico inteiro distinto (todos diferentes entre si). Imaginando nossa lista sendo de tamanho *n*, então, deve ser sorteado números de *-n* e *n*, para cada consulta ao chapéu, e imprimir o número do soldado eliminado a cada rodada e o número do soldado escolhido no final, até que a lista se encontre vazia. Valores negativos fazem a contagem andar para a esquerda a partir do primeiro elemento da lista, enquanto os valores positivos andam para a direita. Considere o valor 0 como um valor inválido e realize um novo sorteio neste caso.