Algoritmos e Estrutura de Dados II

1) Observe o código abaixo:

```
void incluiLista(tlista *l, int info) {
    tno *p, *aux, *ant;
    p = (tno*)malloc(sizeof(tno));
    if (!p) printf("\n Nao tem memoria \n");
    else {
        aux = l->com;
        ant = (tno *) NULL;
        while ((aux) && (aux->dado < info)) {
            ant = aux;
            aux = aux->prox;
        }
        p-> dado = info;
        if (!ant) {
          p->prox = l->com;
          l->com = p;
        }
        else {
            ant->prox = p;
            p->prox = aux;
        }
    }}
```

Responda as seguintes perguntas:

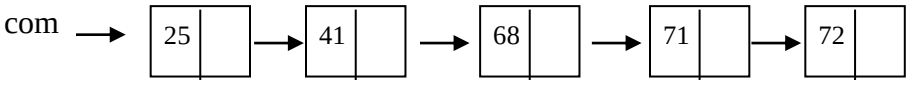
a) No cabeçalho da função incluiLista, a lista l foi passada como referência. Justifique a passagem da lista por referência nesta função.
b) Na atribuição aux = l->com, por que não podemos usar l.com já que com é um campo da lista l?
c) Considere uma lista de inteiros com os seguintes valores de dados: 5,8,12,85. Para cada chamada da função e para cada iteração do loop destacado em vermelho, escreva os valores de ant->dado e aux->dado assumidos durante a execução do algoritmo.

2) Elabore um programa que utilize uma estrutura de dados para manipular uma lista de informações sobre produtos de uma oficina. Para cada produto, deverão ser armazenados os seguintes dados: código (inteiro), índice de periculosidade (R – alto risco ou S – sem periculosidade) e o preço. Os produtos devem ser armazenados de forma ordenada pelo código. No programa, devem ser consideradas as seguintes operações:

a) Incluir um produto (ordenado por código)
b) Excluir um produto com um determinado código
c) Listar todos os produtos cadastrados
d) Retornar a quantidade de produtos com um determinado índice de periculosidade
e) Imprimir as informações de um produto com um determinado código.

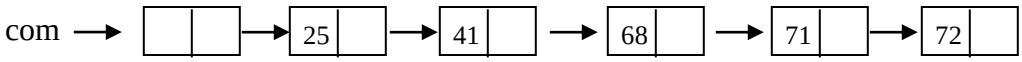
3) Elabore um programa para manipular uma lista de clientes. Para cada cliente, devem ser consideradas as seguintes informações: código, nome (até 100 caracteres), endereço (até 200 caracteres), data ultima compra, data de nascimento. A lista deve ser organizada pelo nome do cliente em ordem crescente.
   a) Dica: a função strcmp (s1, s2) deve ser usada para comparação entre duas cadeias de caracteres: s1 e s2. Os valores de retorno da função são:
      i) 0, se as duas cadeias de caracteres são iguais
      ii) Valor positivo, se o primeiro caractere diferente entre as duas cadeias é maior em s1.
      iii) Valor positivo, se o primeiro caractere diferente entre as duas cadeias é maior em s2.

4) Na literatura, existem várias classificações de listas encadeadas. Uma delas classifica as listas em listas com cabeça e listas sem cabeça. Nas listas encadeadas sem cabeça, o conteúdo do primeiro nó é tão relevante quanto o dos demais. Nesse caso, quando a lista está vazia, não existe nenhum nó e o primeiro nó é NULL. Suponha que L seja uma lista ordenada contendo a idade de 5 pessoas (25,41,68,71,72). L terá a seguinte forma:

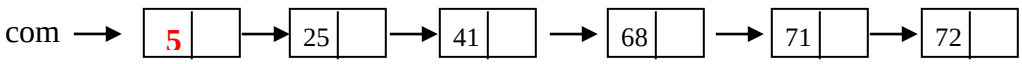


Uma variação deste tipo de lista é a lista encadeada com cabeça. Veja a descrição a seguir:

- Lista encadeada com cabeça: o primeiro nó da lista não contém uma informação relevante para o problema que está sendo resolvido. O primeiro nó serve apenas para marcar o início da lista. O primeiro nó é a *cabeça (=head cell =dummy cell = célula cabeça)* da lista. Neste caso, a Lista sempre deve conter um nó que é a cabeça. Digamos que `com` é o endereço do nó cabeça (*com* é um ponteiro para o nó cabeça). Logo, a lista está vazia se e somente se `com->prox == NULL`. Suponha que L seja a mesma lista do exemplo anterior. L terá a seguinte forma:



Nas listas com cabeça, o nó que representa a cabeça da lista pode ainda conter alguma informação, por exemplo, o tamanho da lista. Neste caso, diz-se que a lista tem um cabeçalho. Considere que L seja uma lista com cabeçalho que deve guardar o tamanho da lista. L teria a seguinte representação:



   a) As operações de incluir, remover e mostrar nós de uma lista que foram analisadas em sala de aula fazem referência a uma lista com ou sem cabeça?

b) Elabore uma nova versão das operações de forma que a cabeça da lista agora seja um cabeçalho contendo o tamanho da lista.

5) Para uma mesma estrutura de dados, podem ser desenvolvidas diferentes estratégias para sua manipulação. Para lista encadeada, por exemplo, existem diferentes variações. Na estratégia usada em sala de aula, a lista foi representada pela struct lista que contém um campo que guarda o endereço do primeiro nó.
struct no {int dado, struct no *prox}; typedef struct no tno;
struct lista { tno * *com,* int *tam*}; typedef struct lista tlista;
// *com* não é um nó
// *com* é um campo da estrutura lista que armazena o endereço do primeiro nó.

Em algumas implementações, a lista é representada por um ponteiro para o primeiro nó somente. Neste caso, a lista não é definida como uma estrutura.
struct no {int dado, struct no *prox}; typedef struct no tno;
tno **com*;
//*com* continua sendo um ponteiro (endereço) para o primeiro nó da lista. Contudo, *com* não é definido como um campo de uma estrutura.

Refaça as operações do TAD Lista encadeada considerando a representação da lista como um ponteiro para o primeiro nó.

6) Fila de prioridade é uma estrutura de dados que mantém uma coleção de elementos, cada um com uma prioridade associada. Existem dois tipos de filas de prioridades: fila de prioridade ascendente e fila de prioridade descendente. Uma fila de prioridade ascendente é um conjunto de itens no qual podem ser inseridos itens arbitrariamente e a partir do qual apenas o menor item pode ser removido. Nesta fila, a operação desenfileirar deve remover o menor item. Já uma fila de prioridade descendente é semelhante, mas só permite a eliminação do maior item. Considerando a descrição acima, use a estrutura lista simplesmente encadeada para criar uma fila ascendente e uma fila descendente. Implemente as operações:
   a) inserirFilaAsc (inserir um elemento na fila ascendente)
   b) inserirFilaDes (inserir um elemento na fila descendente)
   c) removerFilaAsc(remover um elemento na fila ascendente)
   d) removerFilaDes(remover um elemento na fila descendente)

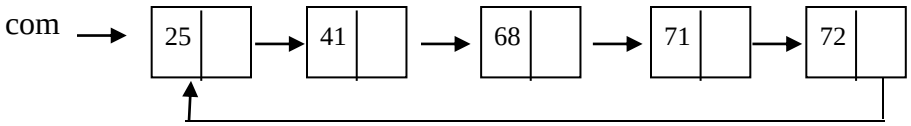
7) Leia o arquivo Lista disponível no SIGAA e discuta sobre as vantagens e desvantagens de cada uma das estratégias de implementação de lista abaixo:
   a) Lista estática com contigüidade física
   b) Lista simplesmente encadeada estática
   c) Lista simplesmente encadeada dinâmica

8) Implemente uma função que receba duas listas simplesmente encadeadas e retorne a lista resultante da concatenação das duas listas recebidas como parâmetro, isto é, após a concatenação, o último elemento da primeira lista deve apontar para o primeiro elemento da segunda lista. Essa função deve obedecer ao protótipo. Utilize a função concatenaLista na função principal para concatenar duas listas com números preenchidos pelo usuário.
   a) concatenaLista(tlista *L1, tlista L2)

9) Exercícios do livro Estruturas de Dados Usando C (Tenenbaum, et al):
   - pág 266 e 267 (4.3.6,4.3.7,4.3.11)

10) Considere as operações de lista duplamente encadeada implementadas em sala de aula e faça as devidas alterações para a seguinte situação:
Na implementação da lista duplamente encadeada, em algumas operações, foram usadas duas variáveis auxiliares (“atual” e “ant”). Por exemplo, na inserção, o novo nó deveria ser inserido entre “atual” e “ant”. Na remoção, “atual” foi definida como a variável que apontava para o nó a ser inserido e “ant” era a variável que guardava o endereço do nó anterior a “atual”. Altere a implementação das operações de forma que não seja utilizada a variável auxiliar “ant”. Esta alteração só é possível porque na lista duplamente encadeada, é sempre possível saber quem é o nó anterior a outro nó.

11) Considere agora uma versão da lista simplesmente encadeada em que o ponteiro prox do último não seja NULL, mas aponte para o primeiro nó.
A lista do exercício 4 com a idade de 5 pessoas (25,41,68,71,72). L teria a seguinte forma:



Altere as funções do TAD Lista simplesmente encadeada de maneira que seja implementada uma lista simplesmente encadeada circular.

12) Faça as mesmas alterações no TAD lista duplamente encadeada para que seja implementada uma lista duplamente encadeada circular. Nesse caso, além do prox do último nó apontar para o primeiro nó da lista, o ponteiro ant do primeiro nó aponte para o último nó da lista.

13) O problema de Josephus é um clássico na computação e já foi reescrito de várias maneiras. Uma das versões é descrita a seguir: 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. 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, 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.
Analise o problema e escolha uma estrutura de dados apropriada para a solução. Observe que na estratégia de eliminação a lista de soldados pode ser percorrida várias vezes de forma circular. Para simplificar o problema considere que o soldado sorteado para iniciar a contagem foi o primeiro.

14) Escreva um programa que tenha uma lista cujos elementos possuem um campo inteiro representando sua prioridade. Quanto menor o valor deste campo, maior a prioridade do elemento. Insira n elementos com prioridades diversas na lista e depois divida a lista em duas, uma com elementos cuja prioridade é menor ou igual ao valor N fornecido pelo usuário e outra com os elementos restantes.

15) Exercício 4.2.3 do livro Estrutura de Dados usando C (Tenenbaun et al, 2005), pág 244, capítulo 4.
Implemente dada uma das operações abaixo nas listas duplamente encadeada e simplesmente encadeada. Considere uma lista de números inteiros positivos.
a) Incluir um elemento no final da lista.
b) Concatenar duas listas.
c) Liberar todos os nós de uma lista.
d) Inverter uma lista de modo que o último elemento se torne o primeiro, e assim por diante.
e) Eliminar o último elemento da lista.
f) Eliminar o enésimo elemento da lista.
g) Combinar duas listas ordenadas em uma única lista ordenada.
h) Formar uma lista contendo a união dos elementos de duas listas (sem repetição).
i) Formar uma lista contendo a intercessão dos elementos de duas listas.
j) Inserir um elemento depois do enésimo elemento da lista.
k) Retornar a soma dos inteiros de uma lista.
l) Retornar a quantidade de elementos de uma lista.
m) Deslocar um nó n posições à frente numa lista (o nó e a quantidade de posições devem ser parâmetros)
n) Criar uma cópia de uma lista.
o) Buscar um elemento em uma lista (retornar 1 se achou e 0, caso contrário).
p) Retornar o nó, sem retirá-lo da lista, que está em uma determinada posição (retornar o nó e não o valor do dado armazenado no nó)
q) Retornar o nó, sem retirá-lo da lista, que contém um determinado dado.