Programação 2024/25

LEI, LEI-PL, LEI-CE

Aula Laboratorial 9

Bibliografia:

K. N. King. *C programming: A Modern Approach* (2nd Edition). W. W. Norton: capítulo 17.

Código de apoio para a aula: https://gist.github.com/FranciscoBPereira

Estruturas Dinâmicas

Exercícios Obrigatórios

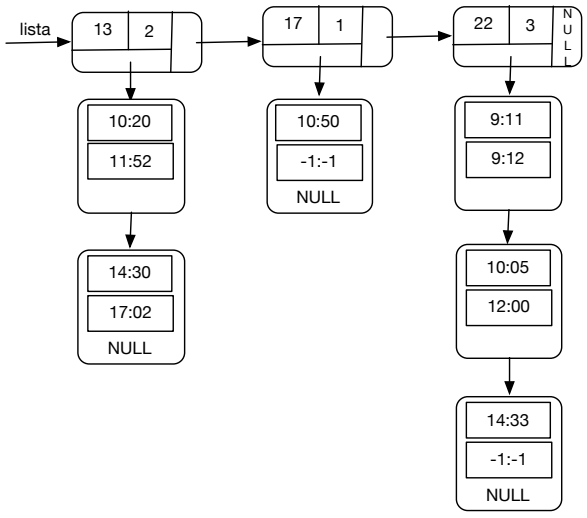
1. Considere as seguintes definições:

```
typedef struct tipoA cliente, *pCliente;
typedef struct tipoB acesso, *pAcesso;
typedef struct {int h, m;} hora;

struct tipoA{
      int id;                 // Identificador único
      int contador;           // Número de utilizações nesse dia
      pAcesso lista;          // Ponteiro para a lista de acessos
      pCliente prox;          // Ponteiro para o próximo cliente
};

struct tipoB{
      hora in, out;           // Horas de entrada e saída
      pAcesso prox;
};
```

Um parque de estacionamento armazena numa estrutura dinâmica informação sobre a utilização dos seus serviços por parte dos clientes registados. **Esta informação diz respeito a um único dia.** Existe uma lista ligada principal com estruturas do tipo *cliente* contendo informação sobre os clientes registados no parque, ordenada pelo valor do campo *id*. De cada nó da lista principal sai uma lista secundária com elementos do tipo *acesso* contendo informação sobre os acessos desse cliente ao longo do dia. Cada nó de acesso regista a hora em que se processou a entrada e a posterior saída do parque. As listas secundárias estão ordenadas por hora de utilização, ou seja, novas utilizações estão sempre no final desta lista.



Caso o cliente ainda se encontre no parque, a hora de saída do último nó tem o valor -1:-1. Estas são consideradas utilizações em aberto. O código disponibilizado cria uma estrutura dinâmica igual à que surge na figura. A partir desse ponto deve implementar e testar as seguintes funções:

a) Escreva uma função em C que indique quantos clientes estão nesse momento dentro do parque. A função recebe como parâmetro um ponteiro para o início da estrutura dinâmica e devolve o valor contabilizado.

b) Escreva uma função em C que contabilize o número de clientes que estavam dentro do parque num determinado momento. A função recebe como parâmetros um ponteiro para o início da estrutura dinâmica e a hora a considerar. Devolve o valor contabilizado.

c) Escreva uma função em C que descubra qual o cliente que já passou mais minutos no parque no dia atual. Na contabilização não devem ser consideradas utilizações em aberto, ou seja, não devem ser consideradas as utilizações em que o cliente ainda se encontre no parque. A função recebe como parâmetros um ponteiro para o início da estrutura dinâmica e o endereço de uma variável inteira onde deve colocar o número de minutos contabilizado. Devolve o *id* do cliente identificado como resultado. Caso não exista nenhum cliente com utilizações completas, a função devolve -1.

d) Escreva uma função em C que elimine um utilizador da estrutura dinâmica. Esta função deve eliminar o nó da lista ligada principal do utilizador especificado e toda a lista secundária contendo os seus acessos. A função recebe como parâmetros um ponteiro para o início da estrutura dinâmica e o id do cliente a eliminar. Devolve um ponteiro para o início da estrutura dinâmica atualizada.

e) Escreva uma função em C que elimine todas as utilizações em aberto que se encontrem nas listas secundárias da estrutura dinâmica. Os espaços retirados da estrutura dinâmica devem ser eliminados e devem ser atualizados os contadores da lista principal. Caso um utilizador fique sem acessos registados, deve ser eliminado da lista principal. A função recebe como parâmetro um ponteiro para o início da estrutura dinâmica. Devolve um ponteiro para o início da estrutura dinâmica atualizada.

f) A função seguinte processa uma ativação de cancela de acesso ao parque:

```
pCliente acessoParque(pCliente lista, int id, hora x);
```

Indica que o cliente com identificador *id* ativou a cancela na hora *x*. Pode corresponder a uma entrada ou a uma saída e pode ser a primeira vez (ou não) que o cliente usa o parque nesse dia.

A função recebe um ponteiro para o início da estrutura dinâmica como um dos seus parâmetros e deve fazer a atualização necessária:

i. Coloca a hora de saída se o cliente *id* tiver a última utilização em aberto.
ii. Adiciona uma nova utilização ao final da lista de acessos do cliente *id,* especificando a sua hora de entrada.
iii. Adiciona o cliente *id* à lista principal, criando o seu primeiro acesso do dia e especificando a hora de entrada.

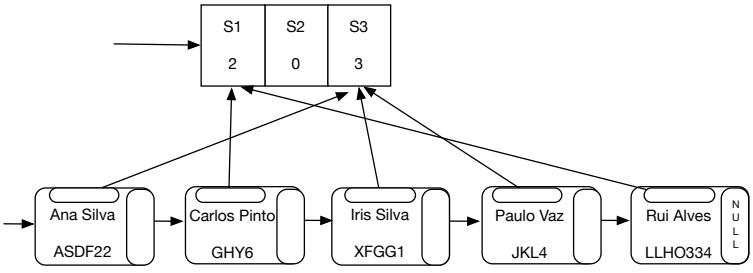
Devolve a estrutura dinâmica atualizada.

2. Considere as seguintes definições:

```
struct sala{
    char id[10];
    int total;
};

typedef struct pessoa no, *pno;
struct pessoa{
    char nome[100], num[20]; // Nome e número do aluno
    struct sala* p;    // Ptr. para a sala onde o aluno fará o teste
    pno prox;
};
```

A informação dos alunos que vão realizar um teste está armazenada numa lista ligada simples, constituída por nós do tipo *no*. Esta lista está organizada alfabeticamente pelo nome do aluno. Existe ainda um array dinâmico, constituído por estruturas do tipo *struct sala*, contendo informação sobre as salas onde vai ser realizada a prova. Cada estrutura do array contém a identificação da sala e um contador indicando quantos alunos farão a prova nesse local. O array não está ordenado de acordo com nenhum critério. Cada nó da lista ligada (i.e., cada aluno) tem um campo *p*, referenciando a sala a que está associado. No exemplo da figura pode verificar-se que existem 3 salas e 5 alunos, estando o aluno Carlos Pinto associado à sala S1.



a) Escreva uma função em C que imprima uma listagem completa dos alunos (nome e número) que vão realizar a prova em cada uma das salas. A listagem deve surgir por sala. A função recebe, como parâmetros, o endereço inicial da lista de alunos, o endereço inicial e a dimensão do array de salas.

b) Escreva uma função em C que imprima na consola o nome e número dos alunos que vão realizar a prova na sala com mais pessoas. Caso exista mais do que uma sala com um número máximo de alunos, a função não escreve nada. A função recebe, como parâmetros, o endereço inicial da lista de alunos, o endereço inicial e a dimensão do array de salas.

c) Escreva uma função em C que verifique se 2 alunos vão realizar a prova na mesma sala. A função recebe, como parâmetros, os nomes dos alunos, o endereço inicial da lista de alunos, o endereço inicial e a dimensão do array de salas. A função devolve 1 se os 2 alunos estiverem na mesma sala, 0 se estiverem em salas diferentes, ou -1, se algum dos alunos não estiver na lista.

d) Escreva uma função em C que transfira um aluno de uma sala para outra. A função tem o seguinte protótipo:

```
struct sala* transfereAl(pno lista, struct sala *s, int *tot,
                         char *nome, char *nSala):
```

A função recebe o início da lista ligada, o endereço inicial e o endereço de uma variável inteira contendo a dimensão do array de salas, o nome do aluno a transferir e o id da sala para onde o aluno deve ser transferido. Se a sala de onde o aluno sair ficar vazia, a função deve retirar esse elemento do array. A função devolve um ponteiro para o início do array de salas depois da atualização. Se a nova sala que for indicada para o aluno não existir, não é efetuada nenhuma alteração na estrutura dinâmica.

e) Escreva uma função em C que adicione um aluno à estrutura dinâmica e o associe a uma das salas. A função tem o seguinte protótipo:

```
pno adicionaAl(pno lista, struct sala *s, int tot, char *nome,
                                  char *num);
```

A função recebe o início da lista ligada, o endereço inicial e a dimensão do array de salas e os dados do novo aluno. Deve adicionar o aluno à lista ligada (mantendo a ordem alfabética) e associá-lo à sala que tenha menos alunos nessa altura. Caso existam várias salas com um número mínimo de alunos, a função deve escolher uma delas. A função devolve um ponteiro

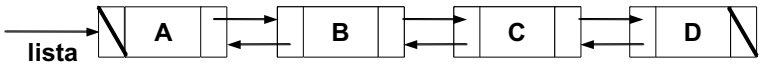
para o início da lista depois da atualização. Ao responder a esta pergunta, pode assumir que o array dinâmico tem, no mínimo, 1 sala.

Exercícios Complementares

3. Considere as seguintes definições:

```
typedef struct dados no, *pno;
struct dados{
      int val;
      pno prev, prox;
};
```

A estrutura `struct dados` permite criar uma **lista duplamente ligada**. Numa estrutura dinâmica deste tipo, cada nó tem dois ponteiros: um (`prev`) aponta para o elemento que está imediatamente antes e o outro (`prox`) aponta para o elemento que está imediatamente a seguir.



O ponteiro `lista` (habitualmente uma variável local da função `main()`) aponta para o primeiro elemento da lista duplamente ligada. Considerando que a informação na lista deve estar ordenada, de forma crescente, pelo valor armazenado no campo `val`, escreva funções em C que efetuem as seguintes operações:

a) Adicionar um novo elemento à lista. A função recebe como parâmetros um ponteiro para o início da lista e o novo valor a adicionar. A lista não deve conter valores repetidos, ou seja, se o valor indicado já estiver armazenado num dos nós, a função não efetua a inserção. Devolve um ponteiro para o início da lista modificada.

b) Mostrar uma listagem completa dos valores armazenados na lista. A função recebe como parâmetro um ponteiro para o início da lista.

c) Mostrar uma listagem completa invertida (do último para o primeiro) dos valores armazenados na lista. A função recebe como prâmetro um ponteiro para o início da lista.

d) Eliminar um elemento da lista. A função recebe como parâmetros um ponteiro para o início da lista e o valor a eliminar. Devolve um ponteiro para o início da lista modificada.

e) Apresentar os elementos que sejam iguais à média dos seus vizinhos (os vizinhos de um nó são os elementos que se encontram imediatamente antes e depois dele). A função recebe como parâmetro um ponteiro para o início da lista.

4. Uma **lista circular** pode ser vista como uma lista ligada em que o último elemento aponta para o primeiro (embora neste tipo de listas não façam muito sentido as referências ao primeiro e último elementos). Considere que se pretende representar através de uma lista circular um conjunto de pessoas que estão sentadas à volta de uma mesa redonda. Cada nó da lista terá um campo para armazenar o nome da pessoa e um ponteiro para o elemento relativo à pessoa que está à sua esquerda.

a) Efetue as definições de tipo necessárias de modo a ser possível implementar a estrutura de dados descrita.

b) Escreva uma função em C que devolva o número de pessoas que estão à volta da mesa. A função recebe como argumento um ponteiro para um dos elementos da lista.

c) Escreva uma função em C que devolva o número de lugares que existem entre duas pessoas. A função recebe como argumentos 2 ponteiros (um para cada uma das pessoas a considerar).

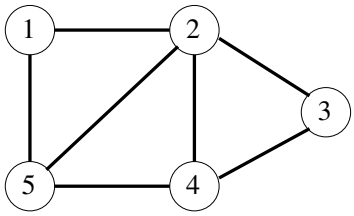
d) Escreva uma função em C que permita sentar a primeira pessoa à mesa. A função recebe como argumento o nome dessa pessoa e, no final, devolve um ponteiro para o nó da lista que acabou de ser inserido.

e) Escreva uma função em C que permita inserir uma nova pessoa na lista circular. A função recebe como argumentos, um ponteiro para um dos elementos da lista, o nome da nova pessoa e o nome da pessoa que vai ficar sentada à sua esquerda (alguém que já deve fazer parte da lista). Se essa pessoa não existir, a inserção fica sem efeito.

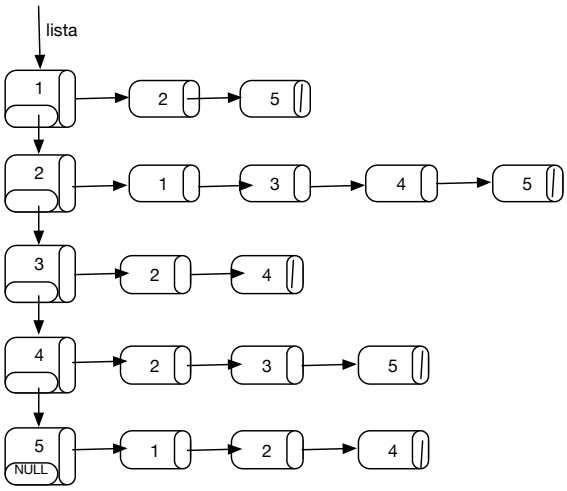
f) Escreva uma função em C que permita mudar alguém de lugar. A função recebe como argumentos, um ponteiro para um dos elementos da lista, o nome da pessoa que deseja mudar de lugar e o seu deslocamento (número de posições que essa pessoa deseja ir para a esquerda).

Trabalho Autónomo

5. Um grafo é uma estrutura de dados não linear constituída por um conjunto e vértices e por várias arestas, cada uma delas ligando 2 vértices. Devido à sua versatilidade são usados em inúmeros problemas reais em que a informação é naturalmente representada em rede, como redes elétricas, redes de computadores, redes sociais, entre muitas outras possibilidades. A seguir é mostrado um pequeno exemplo de um grafo, contendo 5 vértices e 7 arestas. Neste exemplo iremos considerar grafos não dirigidos, ou seja, as arestas que efetuam a ligação entre 2 vértices podem ser percorridas nos 2 sentidos.



Os grafos podem ser armazenados de várias formas. Uma das mais comuns é uma lista de adjacências em que se especifica, para cada vértice do grafo, uma lista contendo os vértices que são adjacentes. A seguir pode ser consultada uma figura ilustrando o grafo anterior no formato de lista de adjacências:



A lista de adjacências de um grafo pode ser representada como uma estrutura dinâmica, neste caso como uma lista de listas. A lista principal deve conter um nó para cada um dos vértices do grafo. De cada um destes nós sairá uma lista secundária contendo a informação sobre as adjacências desse vértice.

a) Efetue as definições de tipos necessárias de modo a ser possível implementar a estrutura de dados descrita. Considere que a única informação armazenada em cada vértice é um valor inteiro.

b) Implemente funções que efetuem algumas das operações básicas de manipulação de grafos:
   i. Listar grafo: Apresentar informação completa sobre o grafo, incluindo os vértices e as arestas existentes.
   ii. Adicionar vértice: Adicionar um novo vértice ao grafo. Nesta altura o vértice ficará sem nenhuma ligação (aresta).
   iii. Adicionar aresta: Adicionar uma nova aresta entre 2 vértices do grafo.
   iv. Adjacente: Verifica se existe uma aresta entre 2 vértices do grafo.
   v. Listar vizinhos: Apresenta todos os vértices vizinhos de um determinado vértice. Considera-se que 2 vértices são vizinhos se tiverem uma aresta a efetuar a ligação direta.
   vi. Remover aresta: Eliminar uma aresta entre 2 vértices.
   vii. Remover vértice: Eliminar um vértice do grafo. Todas as arestas que estejam ligadas a esse vértice devem ser também eliminadas.