Programação

Licenciatura em Engenharia Informática: 1º ano - 2º semestre 2019/2020

Guião laboratorial n.º 4

Estruturas Dinâmicas

Tópicos da matéria:

- Listas ligadas e outras estruturas dinâmicas
- Operações com estruturas dinâmicas

Bibliografia:

K. N. King, *C Programming: A Modern Approach*: Capítulo 17.

A – Listas Ligadas Simples

1. Pretende-se utilizar uma lista ligada simples para armazenar informação sobre o índice de massa corporal de um conjunto de indivíduos. A estrutura a utilizar para armazenar a informação de cada pessoa é a seguinte:

```
typedef struct pessoa no, *pno;
struct pessoa{
      char nome[100];
      int id;
      float peso, altura;
      pno prox;
};
```

O índice de massa corporal é obtido através da seguinte expressão: $imc = peso/altura^2$, em que o peso é especificado em kg e a altura em metros.

a) Crie um ponteiro de lista na função *main()*.

b) Escreva uma função que adicione os dados de uma nova pessoa à lista. O nome, identificador numérico, peso e altura dessa pessoa devem ser obtidos do utilizador. A função deve garantir que o id indicado é único e não se encontra atribuído a outra pessoa que já esteja na lista. A nova pessoa deve ser adicionada ao final da lista ligada.

c) Escreva uma função que mostre na consola a informação completa de todas as pessoas que se encontram na lista ligada, incluindo o seu imc.

d) Escreva uma função que atualize o peso de uma das pessoas que se encontra na lista. O id da pessoa a considerar e o seu novo peso são 2 dos parâmetros da função.

e) Escreva uma função que elimine uma pessoa da lista ligada. O id da pessoa a eliminar é um dos parâmetros da função. O nó retirado da lista deve ser libertado.

f) Escreva uma função que elimine todos os nós da lista ligada.

g) Escreva uma função que elimine da lista todas as pessoas com um imc superior a um determinado limite. O valor limite a considerar é um dos parâmetros da função. Todos os nós retirados da lista devem ser libertados.

2. Pretende-se armazenar a informação da lista ligada construída na questão anterior num ficheiro de texto.

a) Escreva uma função que grave, num ficheiro de texto, a informação das pessoas que se encontram na lista. A função recebe, como parâmetros, o nome do ficheiro e um ponteiro para o início da lista.

b) Escreva uma função que construa uma lista ligada com as pessoas cuja informação está armazenada num ficheiro de texto. O formato do ficheiro é igual ao que foi definido na alínea anterior. A função recebe o nome do ficheiro como parâmetro e devolve a lista construída como resultado.

3. Pretende-se alterar a gestão da lista ligada definida na questão 1, de forma a que as pessoas na lista passem a estar ordenadas por imc crescente. Altere o que for necessário nas funções implementadas na questão 1, para garantir que a informação passa a estar organizada desta forma.

4. Um consultório médico pretende controlar o atendimento de sucessivos pacientes de forma a respeitar a ordem de chegada. Pretende-se implementar um programa que faça a gestão de uma lista ligada em que cada nó corresponde a um dos pacientes que se encontra à espera. A lista ligada funcionará como uma fila de espera em que o primeiro a chegar será o primeiro a ser atendido (estrutura de dados do tipo FIFO, i.e., *First In First Out*).

a) Considerando que cada paciente é completamente identificado pelo seu nome, defina a(s) estrutura(s) que considerar necessária(s) para a resolução do problema.

b) Desenvolva uma função que **adicione um novo paciente ao final da fila de espera**. O nome do paciente que acabou de chegar deve ser introduzido pelo utilizador.

c) Desenvolva uma função que permita **visualizar toda a fila de espera**.

d) Desenvolva uma função que implemente o atendimento de um novo paciente. A função deve **retirar o elemento que está no início da fila de espera** e escrever o seu nome no monitor.

e) Desenvolva uma função que permita retirar da fila um paciente que tenha desistido de esperar. A função deve obter o nome do paciente através do utilizador. Esta operação corresponde a **eliminar um dos nós da lista.**

5. Considerando o exercício anterior, nem todos os pacientes necessitam de ser atendidos com a mesma urgência. Considere que existem três graus de urgência: grávidas (nível 1), idosos e crianças (nível 2) e restantes pacientes (nível 3). Só serão atendidos pacientes do nível 2 se não existirem pacientes do nível 1 e só serão atendidos pacientes do nível 3 se não existirem pacientes dos níveis 1 e 2. Pretende-se que modifique a função de inserção de um novo paciente de modo a ter em conta estes três níveis de prioridade. A fila de espera deve continuar a ser única e a função de atendimento não deve ser alterada (o próximo paciente a ser atendido é o que se encontra no início da fila). Para responder a esta questão deve efetuar as alterações necessárias na definição da(s) estrutura(s) de dados.

Sugestão: Para encontrar o ponto de inserção de um novo paciente, a função deve procurar o primeiro nó da lista que tenha urgência menor (por exemplo, se o novo paciente for de nível 1, então a função deve procurar o primeiro nó que tenha nível 2 ou superior). A inserção do novo paciente é feita imediatamente antes deste ponto.

6. Complete o exercício anterior com 2 funções que permitam armazenar/recuperar a informação da lista ligada em ficheiro. As 2 funções devem realizar as seguintes operações:

- **Gravar:** A informação completa de todos os elementos da lista deve ser gravada num ficheiro de texto. O nome do ficheiro é passado por argumento e o espaço ocupado pela lista deve ser libertado.
- **Recuperar:** A função acede ao ficheiro cujo nome é passado por argumento e reconstrói uma lista ligada com a informação aí armazenada. Devolve como resultado um ponteio para o início da lista que foi construída.

7. Considere a seguinte definição:

```
struct aluno{
      char nome[50];
      int notas[5];
};
```

a) Desenvolva uma função que obtenha um conjunto de dados do utilizador (nomes de alunos e respetivas notas) e os grave num ficheiro binário. Os dados relativos a cada aluno devem armazenados em estruturas do tipo definido em cima. Pode assumir que existem sempre 5 notas associadas a cada aluno. A função recebe como argumento o número de alunos e o nome do ficheiro.

b) Desenvolva uma função que leia a informação relativa a um conjunto de alunos de um ficheiro, calcule a média das notas para cada um e insira a informação numa lista ligada. Nesta lista os nós dos diferentes alunos devem ficar ordenados por nome.

Nota: Antes de implementar a função deve definir a estrutura de dados necessária para a lista ligada (nome, 5 notas e média). A função recebe o nome do ficheiro como argumento. Como resultado deve devolver o ponteiro para o início da lista depois de construída.

c) Desenvolva uma função que elimine da lista os alunos com média inferior à média da turma. A função recebe como argumento o ponteiro para o início da lista e devolve o ponteiro para o início da lista modificada. O espaço relativo aos alunos que são eliminados deve ser libertado.

d) Efetue a seguinte alteração na função proposta na alínea anterior. Os alunos a eliminar da lista original devem ser inseridos numa outra lista (a lista vai conter os nós dos alunos com média inferior à média da turma). Esta nova função deve receber como argumento adicional uma referência para o ponteiro para o início da nova lista (i.e., recebe um ponteiro para esse ponteiro).

8. Considere as seguintes definições:

```
typedef struct linha  no, *pno;
struct linha{
      char st[80];
      pno prox;
};
```

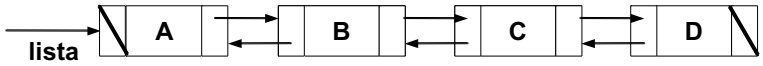
Considere que existem duas listas ligadas distintas constituídas por elementos do tipo `no`. Em cada uma das listas os elementos estão ordenados alfabeticamente (considerando o campo `st`). Pretende-se agrupar os elementos das duas listas numa única lista que deve permanecer ordenada. Escreva uma função que receba os dois ponteiros para o início de cada uma das listas e devolva o ponteiro para o início da lista com todos os elementos. Não deve ser requisitado espaço para os elementos da nova lista (o espaço ocupado pelos nós das duas listas iniciais deve ser aproveitado).

B – Outros Tipos de Listas Ligadas

9. 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` (variável local da função `main()`) aponta para o primeiro elemento da lista duplamente ligada. Considerando que os nós da lista devem estar ordenados pelo valor armazenado no campo `val`, desenvolva funções que efetuem as seguintes operações:

a) Adicionar um novo elemento à lista. A função recebe como argumentos um ponteiro para o início da lista e o novo valor a adicionar. Devolve um ponteiro para o início da lista modificada.

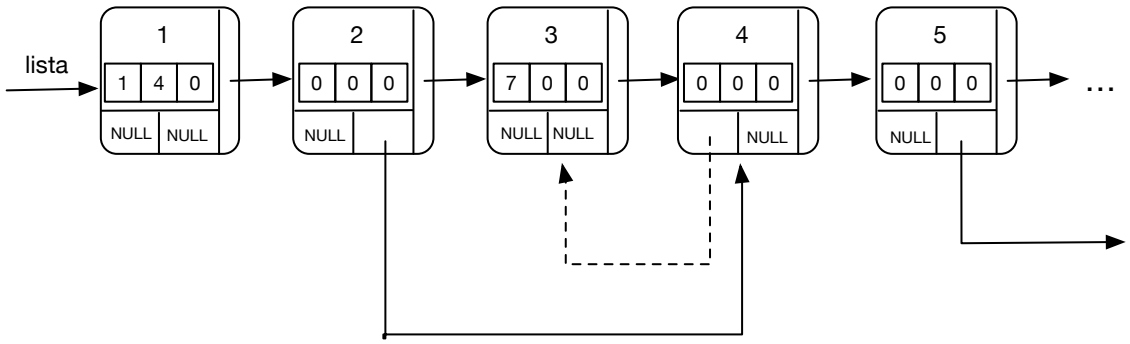
b) Apresentar uma listagem completa dos valores armazenados na lista. A função recebe como argumento um ponteiro para o início da lista

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

d) 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 argumento um ponteiro para o início da lista.

10. Uma estrutura dinâmica armazena um jogo de tabuleiro do tipo jogo da glória. Neste jogo, vários jogadores (3, no máximo) lançam alternadamente um dado para ir avançando nas casas de um tabuleiro. Ganha quem atingir primeiro a última posição. Em algumas casas existem escadas: um jogador que atinja uma escada durante uma jogada, avança para uma casa mais à frente. Em outras posições existem cobras que fazem um jogador recuar algumas casas. Nenhuma posição tem simultaneamente uma escada e uma cobra e não existem cobras nem escadas nas primeira e última posições do tabuleiro. A estrutura seguinte define uma posição, i.e., uma casa do tabuleiro, e permite criar a estrutura dinâmica completa para jogar o jogo:

```
typedef struct posicao no, *pno;
struct posicao{
    int index;    // Índice da posição do tabuleiro (a 1ª tem índice 1)
    int joga[3];  // Ids dos jogadores que estão nesta posição
    pno cobra, escada;  // cobras e escadas existentes nesta posição
    pno prox;
};
```



A estrutura dinâmica exemplificada em cima mostra as primeiras 5 posições de um tabuleiro. São casas sequenciais identificadas pelo seu índice e, em cada casa, existem 2 ponteiros que assinalam a existência de eventuais cobras ou escadas. Se algum destes ponteiros estiver a NULL, não existe ligação deste tipo. Caso contrário apontam para a posição para onde a cobra ou escada levam. No exemplo da figura, a posição 2 tem uma escada para a posição 4. Por sua vez a posição 4 tem uma cobra para a posição 3. Em cada uma das posições do tabuleiro, a tabela *joga* assinala que jogadores se encontram nesse local. Cada jogador tem um identificador inteiro positivo para esse efeito. No exemplo, os jogadores 1 e 4 estão na posição 1 e o jogador 7 está na posição 3. A tabela *joga* serve apenas para identificar quais os jogadores que se encontram numa determinada posição, pelo que a ordem pela qual os identificadores estão armazenados não é relevante ({1, 4, 0} é equivalente a {4, 0, 1}).

a) Escreva uma função em C que mostre em que posição se encontram os jogadores no tabuleiro. A função recebe como argumento um ponteiro para o início da estrutura dinâmica

b) Desenvolva uma função em C efetue uma jogada. A função tem o seguinte protótipo:

```
int jogada(pno lista, int totPos, int idJog, int dado);
```

Recebe um ponteiro para o início da estrutura dinâmica, o número de casas do tabuleiro (o número de nós da lista), a identificação do jogador que está a fazer a jogada e o valor que saiu no dado. Deve atualizar a estrutura dinâmica movendo o jogador para a nova posição. Devolve a casa em que ele ficar depois da jogada. Se o jogador indicado não existir, a função devolve 0.
Caso a jogada o faça ultrapassar o limite do tabuleiro, deve ficar colocado na última casa desse mesmo tabuleiro. Por exemplo: se o jogador 4 tiver 1 no dado, deve sair da posição 1 e terminar a jogada na posição 3.

c) Desenvolva uma função em C que faça uma cópia de uma estrutura dinâmica deste tipo. A função recebe um ponteiro para o início da estrutura dinâmica e o número de casas do tabuleiro. Devolve um ponteiro para uma nova estrutura dinâmica que seja uma cópia da recebida por parâmetro. A estrutura dinâmica original não deve ser modificada.

11. 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 ao último elemento).
Considere que se pretende representar através de uma lista circular um conjunto de pessoas que estão 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 necessárias de modo a ser possível implementar a estrutura de dados descrita.

b) Desenvolva uma função 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) Desenvolva uma função 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) Desenvolva uma função 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) Desenvolva uma função 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.

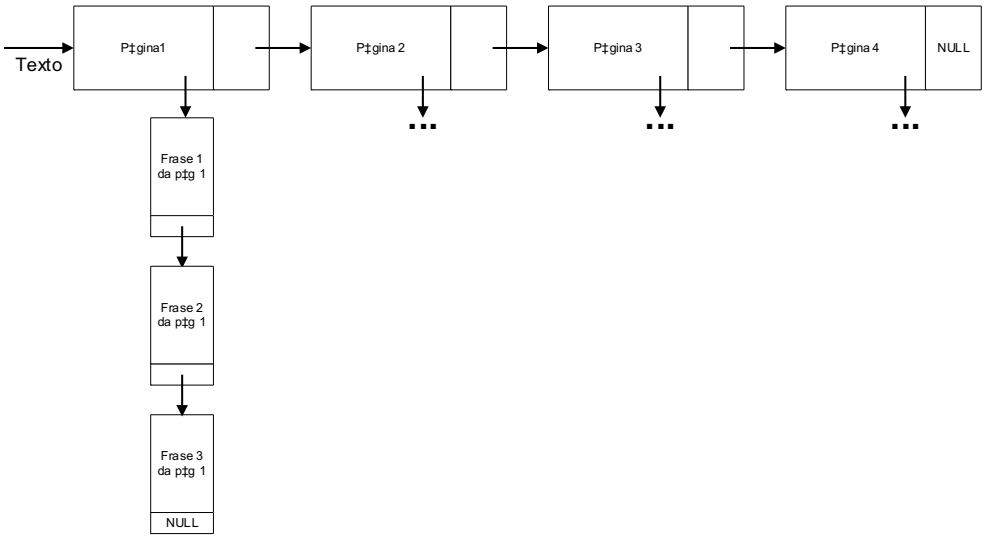
f) Desenvolva uma função que permita mudar alguém de lugar. A função recebe como argumentos, um ponteiro para um dos elementos da lista, o nome da pessoa que deseja mudar de lugar e o seu deslocamento (número de posições que essa pessoa deseja ir para a esquerda).

12. Pretende-se encontrar uma representação dinâmica para um texto. Considere que o texto está dividido em páginas e que, em cada uma dessas páginas, existem frases. Qualquer frase é formada por um conjunto de palavras separadas por um ou mais espaços.

Uma possível representação dinâmica para um texto é a seguinte:

- uma lista ligada em que cada nó representa uma página. Cada um destes nós tem a seguinte informação:
  - número da página;
  - número de frases existentes nessa página;
  - frases que fazem parte da página;

Para representar a terceira componente de cada uma das páginas (as suas frases) é conveniente utilizar uma (nova) lista ligada. Cada nó da lista principal terá então um ponteiro para uma nova lista onde estão armazenadas as várias frases dessa página.
A estrutura completa terá o seguinte formato e pode ser designada como uma lista de listas:



Neste exemplo o texto tem 4 páginas e a primeira página tem 3 frases. Um texto é referenciado pelo ponteiro para o primeiro elemento da lista principal. A estruturas relativas a cada uma das páginas foram inseridas por ordem crescente na lista ligada.

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

b) Desenvolva uma função que escreva no monitor uma determinada frase. A função recebe como argumentos o ponteiro para o início da lista principal, o número da página onde se encontra a frase e a sua posição nessa página.

c) Desenvolva uma função que procure uma determinada palavra no texto. A função recebe como argumentos o ponteiro para o início da lista principal e uma palavra. Deve escrever no monitor as frases em que esta palavra se encontra.

d) Desenvolva uma função que adicione uma frase a uma determinada página. A função recebe como argumentos o ponteiro para o início da lista principal, o ponteiro para o início da frase a adicionar e informação sobre o sítio onde a inserção deve ser efetuada (número da página e posição da nova frase nessa página).

e) Desenvolva uma função que adicione uma nova página ao texto. A página deve ficar em branco. A função recebe como argumento o ponteiro para o início da lista principal e a indicação

do sítio onde a nova página vai ser inserida (ou seja, o número da nova página). Se já existir uma página com esse número no texto, a página antiga e todas as seguintes devem ser alteradas (deve ser adicionada uma unidade ao seu número). A função devolve um ponteiro para o início da lista principal.

f) Considere que existe um limite para o número de frases por página (este limite é definido através de uma constante do programa). Altere a função proposta na alínea d) de modo a considerar esta limitação. Se, ao introduzir uma nova frase o limite de frases for ultrapassado, a última frase da página deve passar para o início da seguinte.

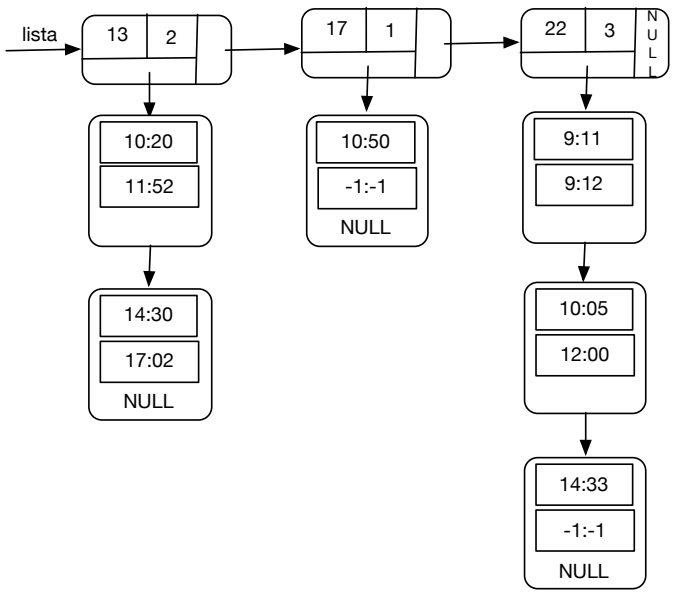
13. Considere as seguintes definições:

```
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. 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.



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 indique 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 parâmetro um ponteiro para o início da estrutura dinâmica e devolve o identificador numérico do cliente. Caso não exista nenhum cliente com utilizações completas, a função devolve -1.

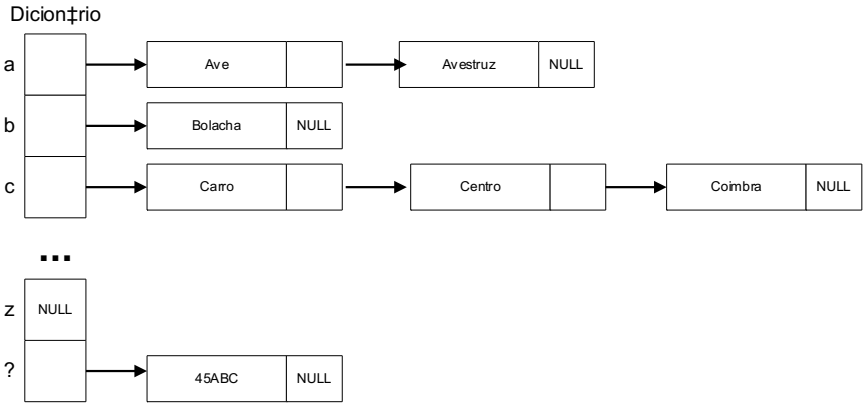
c) Escreva uma função em C que elimine todas as utilizações em aberto que se encontrem nas listas secundárias da estrutura dinâmica. Os espaços retirados da estrutura dinâmica devem ser eliminados e devem ser atualizados os contadores da lista principal. A função recebe como parâmetro um ponteiro para o início da estrutura dinâmica.

d) 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 esse cliente usa o parque nesse dia. A função recebe o ponteiro para o início da estrutura dinâmica como um seus parâmetros e deve fazer a atualização necessária: coloca a hora de saída se este cliente tiver a última utilização em aberto ou adiciona uma nova utilização ao final da lista especificando a sua hora de entrada. Devolve a estrutura dinâmica atualizada.

14. Pretende-se encontrar uma representação dinâmica para um dicionário de palavras que possam surgir num ficheiro de texto.
Uma possibilidade é ter um conjunto de 27 listas ligadas (uma para cada letra do alfabeto mais uma para palavras que não tenham início em caracteres do alfabeto) em que vamos armazenando as palavras que vão surgindo. Cada palavra pertencente ao texto tem um nó numa das listas (a lista correspondente ao primeiro caracter da palavra). Além disso, cada palavra tem associado um contador inteiro que indica quantas vezes ela já surgiu no texto.
Se agruparmos os ponteiros iniciais de cada uma das listas num array de ponteiros o processamento do dicionário torna-se bastante mais simples. A estrutura genérica terá então o seguinte formato.



As listas parciais também estão ordenadas alfabeticamente. Não existe diferença entre caracteres minúsculos e maiúsculos.

a) Defina as estruturas necessárias de modo a poder implementar o dicionário.

b) Desenvolva uma função que receba uma palavra como argumento e devolva como resultado o número de vezes que essa palavra já surgiu no texto.

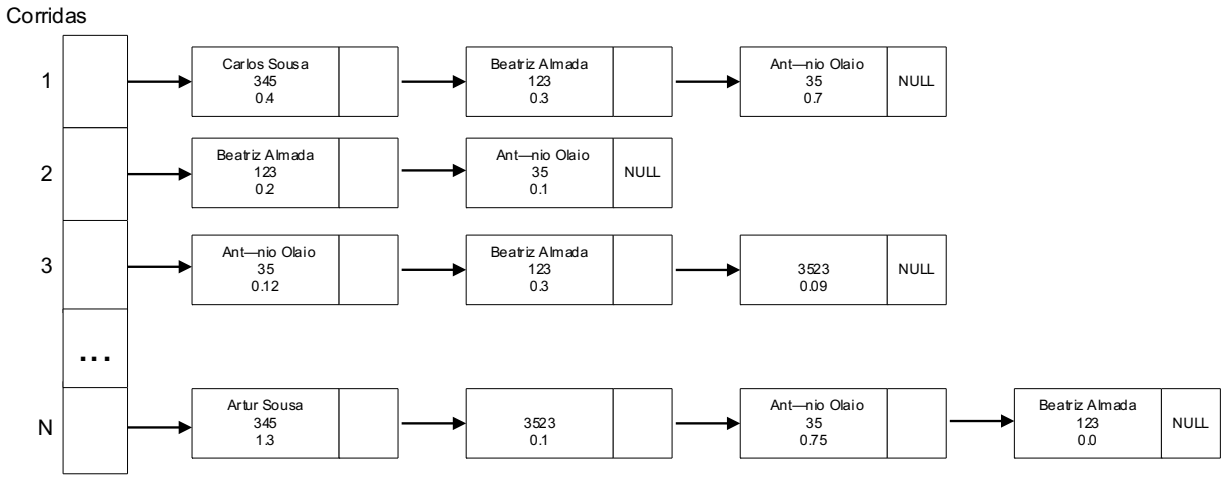
c) Desenvolva uma função que receba uma palavra como argumento e atualize o dicionário. Se essa palavra já fizer parte do dicionário o contador respetivo deve ser atualizado. Caso contrário deve ser criado um novo nó e inserido no sítio respetivo.

d) Desenvolva uma função que crie um dicionário a partir de um texto. A função recebe como argumento o nome do ficheiro de texto a tratar.

e) Desenvolva uma função que armazene o conteúdo do dicionário num ficheiro de texto.

f) Desenvolva uma função que armazene o conteúdo do dicionário num ficheiro binário.

15. Vários alunos do ISEC participaram ao longo do ano nas famosas provas de ciclismo organizadas pela Associação de Estudantes. Um programa para fazer a gestão das classificações em cada uma das provas utiliza uma estrutura dinâmica com o formato exemplificado na figura. Para cada uma das provas realizadas, a informação sobre os concorrentes que a terminaram é armazenada numa lista ligada (existe uma lista para cada prova). A ordem pela qual os alunos aparecem nas listas indica a posição em que eles terminaram cada uma das corridas.
Ao longo do ano foram realizadas N corridas (logo, existem N listas). Este identificador foi definido através da diretiva `#define` no início do módulo (pode assumir que está definido e que pode ser utilizado pelas suas funções).
Os N ponteiros que apontam para o início de cada uma das listas com as classificações, são elementos de um array de ponteiros, de modo a facilitar o processamento da informação sobre as várias corridas.



A informação relativa a cada um dos concorrentes é armazenada numa estrutura com a seguinte definição:

```
typedef struct concorrente no, *pno;
struct concorrente{
      char nome[200];       /*nome do concorrente*/
      int id;               /*n.º de aluno: identificador único*/
      float analise;        /*nível de álcool no sangue*/
      pno prox;
};
```

a) Desenvolva uma função que apresente na consola a classificação verificada em cada uma das N corridas.

b) Desenvolva uma função que crie uma nova lista ligada com informação sobre os atletas que terminaram todas as corridas realizadas. Os elementos dessa nova lista devem ser do tipo `struct concorrente`. O valor do campo `analise` nas estruturas da nova lista é irrelevante. A função recebe como argumento o endereço inicial do array de ponteiros e devolve um ponteiro para o início da lista criada.

Notas:
- as listas originais não devem ser alteradas;
- a ordem dos elementos na nova lista não é relevante.

c) O campo `analise` indica a quantidade de álcool no sangue que existe nos atletas no final da cada corrida. Verifica-se que alguns dos concorrentes ultrapassam o limite máximo admitido. Estes concorrentes devem ser punidos e expulsos das provas de ciclismo organizadas no ISEC. Desenvolva uma função que elimine de todas as listas com as classificações, os concorrentes que ultrapassem em qualquer uma das provas o limite máximo estabelecido. O espaço ocupado pelos nós retirados das listas deve ser libertado. A função recebe como argumentos o endereço inicial do array de ponteiros e o limite máximo admitido.

Exemplo: considerando a figura anterior, se o máximo estabelecido for de 0.5, os atletas António Olaio e Artur Sousa deveriam desaparecer de todas as listas com as classificações das provas.

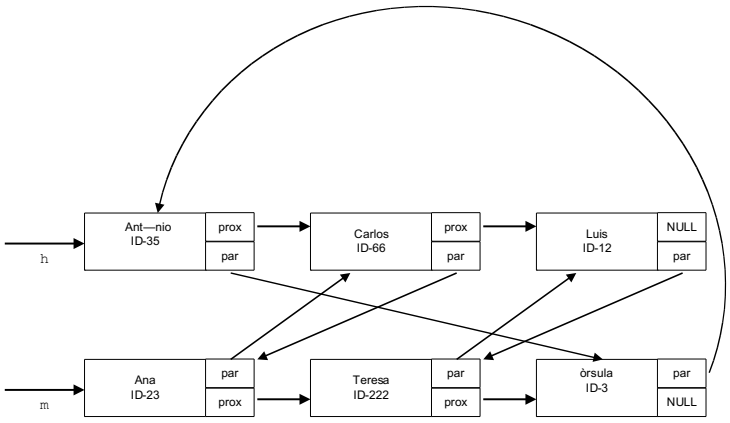
16. Considere as seguintes declarações:

```
typedef struct pessoa aluno, *paluno;

struct pessoa{
      char nome[200];  /* nome  do aluno */
      char id[20];     /* numero de aluno */
      paluno prox;    /*pont para o proximo elemento da lista*/
      paluno par;      /* ponteiro para o par */
};
```

No baile de gala do ISEC, a informação sobre a constituição dos pares de alunos que vão abrir a sessão de dança é guardada em duas listas ligadas. Os elementos de cada uma das listas são do tipo aluno. Numa das listas, cada nó tem informação sobre um aluno rapaz. Na outra lista, cada nó tem informação sobre uma aluna rapariga. Os elementos de cada uma das listas estão ordenados alfabeticamente pelo campo nome. Cada nó pertencente a uma das listas tem dois ponteiros: o ponteiro prox aponta para o próximo elemento da sua lista, enquanto o ponteiro par aponta para o elemento da outra lista que será o seu par no baile de gala. Na figura seguinte pode

observar-se um exemplo. Na situação ilustrada existem 3 alunos em cada uma das listas. Os pares formados são os seguintes: António/Úrsula, Carlos/Ana e Luís/Teresa. Para responder às questões seguintes, pode assumir que não existem alunos sem par em nenhuma das listas.



a) Desenvolva uma função que adicione um novo par de alunos às listas já existentes. A função deve receber, como argumentos, os nomes e os números dos dois alunos e o endereço dos ponteiros que apontam para cada uma das listas de alunos.
O protótipo da função poderá ser o seguinte:

```
void adiciona_par(paluno* lista_h, paluno *lista_m, char *nome_h,
char *id_h, char *nome_m, char *id_m);
```

Os argumentos lista_h e lista_m são, respetivamente, os endereços dos ponteiros para os primeiros elementos da lista de alunos e de alunas. Os argumentos nome_h e id_h contêm informação específica do aluno (o seu nome e o seu identificador) e os argumentos nome_m e id_m contêm a mesma informação relativa à aluna que vai completar o par.
Os dois novos nós inseridos devem ficar ligados através do campo par de cada um deles. As listas devem permanecer ordenadas alfabeticamente.

b) Na sua opinião, porque razão é que os dois primeiros argumentos da função adiciona_par são referências para os ponteiros e não referências para o primeiro elemento de cada uma das listas (i.e., porque é que são ponteiros para ponteiros e não ponteiros para o início das listas)?

c) Desenvolva uma função que guarde num ficheiro a informação armazenada nas duas listas. A única restrição em relação à ordem pela qual a informação deve ser guardada é a de que os dados relativos aos alunos que formam um par devem ficar juntos. A função recebe, como argumentos, o nome do ficheiro e dois ponteiros (para o início de cada uma das listas).
Pode escolher um ficheiro binário ou um ficheiro de texto. O espaço ocupado pelos elementos das listas deve ser libertado.

17. Considere as seguintes declarações:

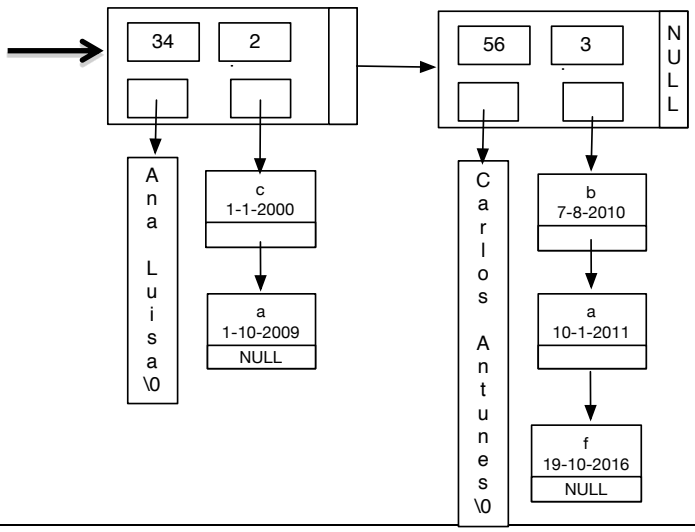
```
typedef struct pessoa pessoa, *ppessoa;
typedef struct resposta resp, *presp;

typedef struct{
    int dia, mes, ano;
    char op;
} info;

struct pessoa{
     char *nome;
     int idade;
     int nresp;
     presp dados;
     ppessoa prox;
};

struct resposta{
     info d;
    presp prox;
};
```

Estas declarações permitem criar uma estrutura dinâmica que armazena informação sobre as respostas dadas a um determinado inquérito. Para cada pessoa que respondeu ao inquérito, a informação inclui o seu nome, idade e respostas já efetuadas. O número de respostas pode variar de pessoa para pessoa. O modo como a informação está organizada é o seguinte: cada nó da lista principal é do tipo *pessoa* e gere a informação relativa a uma pessoa: a sua idade está armazenada no campo *idade* e o seu nome está armazenado num vetor dinâmico que pode ser acedido através do ponteiro *nome* (no final do nome encontra-se o caracter *´\0´*).
As respostas ao inquérito de cada pessoa estão armazenadas numa lista secundária que tem nós do tipo *resp*. De cada nó da lista principal sai uma destas listas contendo informação das respostas dadas por essa pessoa (cada resposta é caracterizada pela opção e pela data). A partir do nó da lista principal, o campo *dados* permite aceder à lista secundária e o campo *nresp* indica quantas respostas já existem nessa lista. Na figura pode ver-se um exemplo de uma lista ligada que armazena informação sobre 2 pessoas que responderam ao inquérito. A Ana Luisa respondeu a duas questões e o Carlos Antunes respondeu a 3.



a) A informação que permite reconstruir uma lista ligada do tipo descrito foi armazenada num ficheiro binário. Pretende-se que desenvolva uma função em C que aceda ao ficheiro e consiga reconstruir a lista. O ficheiro tem o seguinte formato:

- No início encontra-se um inteiro indicando quantos nós tem a lista principal (i.e., quantas pessoas responderam).
- Depois disso, para cada pessoa, a informação guardada é a seguinte: inteiro indicando a sua idade, inteiro indicando quantos caracteres tem o seu nome, sequência de caracteres referente ao seu nome (esta sequência no ficheiro não inclui o '\0'), inteiro indicando quantas respostas já efetuou, campo *info* de cada um dos nós da lista secundária contendo informação sobre as suas respostas (o número de campos *info* guardados é igual ao número de respostas dadas por esta pessoa, i.e., igual ao número de nós da sua lista secundária).
- A informação sobre cada pessoa é guardada sequencialmente, ou seja, primeiro surge toda a informação da primeira pessoa, depois da segunda e assim sucessivamente.

A função recebe o nome do ficheiro como argumento e devolve um ponteiro para o início da lista reconstruída.

b) Desenvolva uma função que adicione uma nova pessoa ao final da lista. Esta função deve adicionar os dados pessoais da pessoa e todas as respostas que ela já efetuou. Toda a informação é indicada pelo utilizador. A função recebe um ponteiro para o início da lista como argumento e devolve um ponteiro para o início da lista depois da inserção.

c) Desenvolva uma função em C que grave a informação da lista num ficheiro binário com o formato especificado na alínea a). A função recebe como argumentos o nome do ficheiro e um ponteiro para o início da lista.