INF1007 - PROGRAMAÇÃO II

LISTA DE EXERCÍCIOS 10

1. Considerando os seguintes tipos estruturados:

```
struct data{
  int dia;
  int mes;
  int ano;
};
typedef struct data Data;

struct paciente{
  char nome[41];
  int idade;
  char tipo_sanguineo[4];      /* "O+", "A+", "B+", "AB+", "O-",
                                  "A-", "B-" ou "AB-" */
};
typedef struct paciente Paciente;

struct doador{
  Paciente dados_paciente;
  Data data_doacao;
};
typedef struct doador Doador;

struct exame{
  char tipo_exame[41];      /* "glicemia", "hemograma",
                               "natremia", "leucopenia",
                               "ferritina" ou "hipocloremia" */
  Paciente dados_paciente;
  struct exame *prox;
};
typedef struct exame Exame;
```

a) Considerando um vetor de ponteiros para `Doador`. Escreva a função `ordenaDoadores` que, utilizando método quick sort, ordena o vetor de ponteiros para `Doador` pelo `tipo_sanguineo` do doador em ordem alfabética crescente. Em caso de mesmo tipo sanguíneo, os doadores devem ser ordenados crescentemente de acordo com `nome` do paciente. Você deve obrigatoriamente escrever uma função auxiliar de comparação. Se você preferir, você pode usar a função `qsort` da biblioteca.

b) Considerando um vetor de ponteiros para `Doador` ordenado em ordem alfabética crescente por `tipo_sanguineo` e usando como critério de desempate o `nome` do paciente em ordem alfabética. Implemente a função `tipoDisponivel` que usando a técnica de busca binaria, encontra e retorna um ponteiro para um `Doador` de um determinado tipo sanguíneo existente nesse vetor. Você deve obrigatoriamente escrever uma função auxiliar de comparação.

c) Considerando uma lista simplesmente encadeada com pedidos de exame, onde os nós encadeados da lista são representados pelo tipo estruturado `Exame`. Escreva a função `removePaciente` que remove da lista encadeada todos os exames de um determinado `paciente` de acordo com o seu `nome`.

d) Considerando um vetor de ponteiros para `Doador`. Implemente a função `criaListaDoadores` que cria e preenche uma lista simplesmente encadeada contendo todos os doadores presentes no vetor de ponteiros para `Doador`. Os nós da lista devem ser representados pela seguinte estrutura:

```
struct nodoador{
  Doador *doador;
  int dias_ultima_doacao;
  struct nodoador *prox;
};
typedef struct nodoador NoDoador;
```

Ao criar os nós da lista, a função também deve calcular a quantidade de dias desde a última doação do doador (de acordo com uma data recebida por parâmetro) e preencher essa informação no campo `dias_ultima_doacao` da estrutura.

e) Com base na lista simplesmente em encadeada criada no exercício anterior, escreva a função recursiva `totalDoadoresTipoRec` que conta e retorna o número total de doadores de um determinado tipo sanguíneo existentes em uma lista encadeada composta por nós do tipo `NoDoador`. Obrigatoriamente a função deve ser implementada de forma recursiva.

2. Um grupo de pesquisa em segurança da informação está estudando a frequência de uso de senhas semelhantes pelos usuários de um determinado sistema. Após processar o banco de dados contendo as senhas dos usuários, o seguinte arquivo contendo 430 mil senhas foi criado:

   http://www.inf.puc-rio.br/~elima/prog2/senhas.zip

   (ATENÇÃO: O arquivo está compactado! Você deve descompacta-lo visualizar o conteúdo!)

   No arquivo que foi criado, cada linha representa uma determinada senha. O primeiro número de cada linha indica o tamanho da senha (número de caracteres) e o segundo número indica a quantidade de ocorrências dessa senha (frequência). Por exemplo, se a senha "vogel1282", de tamanho 9, foi utilizada por 25 usuários, haverá no arquivo uma linha como esta:

   ```
   9 25 vogel1282
   ```

   O seguinte programa faz a leitura do arquivo de senhas e as armazena em um vetor de ponteiros:

   http://www.inf.puc-rio.br/~elima/prog2/senhas.c

   Utilizando o programa base, implemente e teste as seguintes funções:

a) O objetivo inicial do estudo é avaliar somente as senhas com um alto grau de utilização. Para isso, você deve implementar a função `ordenaFrequencias` que ordena o vetor original de forma crescente de acordo com a frequência de utilização das senhas.

b) Para facilitar a analise da frequência de utilização das senhas é necessário sabermos o índice inicial e final de determinadas faixas de frequências no vetor de senhas. Para isso, você deve escrever a função `buscaFaixaFrequencia` que, dada uma certa faixa de frequências, ela disponibiliza os índices de início e fim dos registros desta faixa usando a técnica de busca binária no vetor de ponteiros para senhas ordenado pela frequência. Esta função deve retornar 1 se encontrar ao menos um registro na faixa solicitada e -1 se não for encontrado qualquer registro na faixa solicitada. A função também deve retornar o índice inicial e final da faixa de frequências solicitada.

   Por exemplo, pode-se utilizar essa função para sabermos o índice inicial e o índice final das senhas pertencentes a faixa de frequência que varia entre 25 e 35. A função deve ser capaz de retornar o índice da primeira senha com frequência 25 e o índice da ultima senha com frequência 35 presentes no vetor de senhas.

c) Sabendo que as senhas com a maior quantidade de caracteres são as senhas mais seguras, deseja-se ordenar determinadas faixas de frequências de acordo com o tamanho das senhas. Utilizando a função `buscaFaixaFrequencia` implementada no exercício anterior, crie a função `ordenaFaixaFrequencia` que, dada uma certa faixa de frequências, ela ordena somente as senhas presentes naquela faixa de frequência de acordo com o tamanho das senhas (de modo crescente), mantendo o resto do vetor inalterado. A função deve retornar 1 se encontrar ao menos um registro na faixa solicitada e -1 se não for encontrado qualquer registro na faixa solicitada. A função também deve retornar o índice inicial e final da faixa de frequências ordenada.

f) Uma senha pode ser considerada "não segura" se ela possuir uma quantidade muito pequena de caracteres ou for utilizada por uma quantidade muito grande de usuários. Para classificar as senhas não seguras, você deve criar uma lista simplesmente encadeada contendo todas as senhas com menos de 4 caracteres ou cuja frequência de utilização é maior que 40. Crie a função `naoSeguras` que recebe um vetor de ponteiros para Senhas e retorna uma lista encadeada contendo todas as senhas não seguras existentes. Os nós da lista devem ser representados pela seguinte estrutura:

```
struct nosenha{
  Senha *senha;
  struct nosenha *prox;
};
typedef struct nosenha NoSenha;
```

g) É importante saber o número de usuários que utilizam senhas não seguras. Para isso, implemente a função recursiva `contaUsuariosRec` que recebe como parâmetro uma lista simplesmente encadeada gerada pela função `naoSeguras` e retorne o total de usuários que utilizam senhas não seguras. Lembre-se que a frequência de utilização da senha indica o número de usuários que a utilizam. Obrigatoriamente a função deve ser implementada de forma recursiva.

d) Utilizando as funções implementadas nos exercícios anteriores, modifique a função principal do programa base fornecido de forma que ele permita que o usuário possa digitar uma determinada faixa de frequência e o programa exiba na tela somente as senhas existentes na faixa solicitada e em ordem crescente de acordo com o tamanho das senhas. O programa também deve exibir na tela todas as senhas não seguras existentes juntamente com o total de usuários que utilizam senhas não seguras.