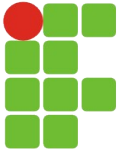


| | | |
|---|--|---|
|  <p>INSTITUTO FEDERAL NORTE DE MINAS GERAIS Campus Montes Claros</p> | INSTITUTO FEDERAL DO NORTE DE MINAS GERAIS | |
| | Aluno(a): | |
| | Curso: Ciência da Computação | |
| | Ano: 2023 | |
| | Professor: Laércio Ives Santos | |
| | Disciplina: Algoritmos e Programação | |
| | Postagem no <i>Class</i> : 09/11/2023 | Apresentação: 10/11/2023 Valor: 30 pontos |

Casa de Apostas

Sua equipe de desenvolvimento foi contratada para criar um sistema para gerir os dados de uma casa de apostas. Tais Apostas são realizadas em partidas de um campeonato de futebol.

- O Apostador (cliente) escolhe quantas apostas quer fazer (vetor dinâmico que deve ser devolvido) pela função ex: 'fazerApostas'
- os dados dos clientes, partidas e apostas devem ser salvos em disco.

1 – Controle de Clientes

```
typedef struct {char cpf[12],
char nome[100],
char telefone[14],
float renda,
char rua[30], char numero[6], char bairro[30], char cidade[30], char estado[2]} Cliente;
```

Funções para clientes: Cadastrar novo Cliente, mostrar todos os clientes, buscar um cliente específico pelo CPF, Alterar os dados de um Cliente.

Seu Controle de Clientes deve manter um vetor contendo uma estrutura de dados de clientes com cpf e posição no arquivo ordenado pelo cpf para permitir a busca binária pelo cpf do cliente.

Esse tipo de estrutura de dados é chamada de índice nos Sistemas Gerenciadores de bases de dados modernos e usam estruturas de dados mais robustas e complexas que somente ordenação.

Sempre que for realizar uma busca de um determinado cliente, a função de busca deve abrir o arquivo contendo a estrutura de dados ('ChaveCliente.bin') realizar a busca na estrutura (descobrendo assim) onde aquele registro está localizado no arquivo principal de clientes ('Clientes.bin') e depois posicionar (com a função fseek) no arquivo 'Clientes.bin' e retornar o registro buscado.

Estrutura proposta:

```
typedef struct {
char cpf_do_Cliente[12],
posicao int // guarda a posição em que o cliente foi inserido.
} ChaveCliente;
```

arquivo ChaveCliente.bin

```
{'1234567', 0}
{'2356756', 3}
{'333567', 4}
{'5612347', 1}
{'6754432', 2}
....
```

Arquivo Cliente.bin

```
{'1234567', 'Carlos Silva', 'Moc'....}
{'5612347', 'Maria Silva', 'Januaria', ....}
{'6754432', 'Ana Carla', 'Moc', ...}
{'2356756', 'José Cardoso', 'Januaria', ...}
{'333567', 'Mariana das Dores', 'moc', ...}
....
```

void NovoCliente();

Essa função é responsável por solicitar que usuário digite os dados de um novo cadastro de clientes inserir os dados no arquivo 'Clientes.bin' e atualizar o vetor contendo a estrutura de dados 'ChaveCliente'.

O cpf não pode conter valores repetidos, sendo assim seu programa deve ter uma função de busca que faça tal controle.

Void mostrarTodos();

Simplesmente lista os clientes cadastrados no sistema.

int buscarClientes(char *cpf, int tamanho, ChaveCliente *vet);

recebe uma string que será o CPF do Cliente que se quer buscar, a quantidade de clientes cadastrados e o vetor contendo a estrutura de dados (cpf, posição).

Realiza busca binária no vetor vet, caso encontre aquele cpf retorna a posição do valor no vetor. caso contrário retorna -1.

void alterarCliente();

Lê o cpf do cliente que se quer alterar, utiliza a função 'buscarClientes()' para saber onde aquele cliente está posicionado no arquivo de clientes, mostra os dados ao usuário (caso encontre), verifica quais dados o usuário quer alterar, lê os novos dados e salva no vetor. Essa opção não pode dar ao usuário a possibilidade de alterar o cpf pois o mesmo é a chave de busca e uma alteração do CPF acarreta uma mudança na ordenação do vetor de estruturas.

2 - Partidas

Partida{Identificador da partida (incrementado automaticamente), data da partida, time A, time B, estádio onde ela foi realizada, gols time A, gols time B, terminada? (0, 1) }

Funções para partida: Nova Partida, Finalizar partida, ver partidas em uma data específica ou de um time específico, ou que um time específico venceu.

void NovaPartida();

Cadastra uma nova partida lendo os dados do usuário e salvando no arquivo.

void FinalizarPartida(char *dataPartida);

Recebe uma data específica, busca por partidas naquela determinada data, pergunta ao usuário ‘quais partidas ele quer finalizar’ (ou seja, alterar o valor do membro 'terminada' para 1)... altera a salva no arquivo (partidas.bin).

Seu sistema deve permitir ainda que o usuário possa visualizar os dados das partidas realizadas em uma determinada data, ou por um determinado time ou mesmo as partidas que um time específico não venceu.

3 - Apostas

```
typedef struct{
char cpf_do_Cliente[12],
char data[11],
float valor_aposta,
int Identificador_da_partida,
char timeApostado, // time em que apostou
int vencedor,
float cotacao
} Aposta;
```

Inserir nova aposta.

void InserirAposta();

Ler dados da aposta e salvar em disco.

Não se pode inserir clientes que não estejam previamente cadastrados. Seu sistema deve buscar pelo cliente e verificar se ele está cadastrado.

Sua função também deve mostrar uma lista de partidas de uma determinada data (a data é digitada pelo usuário) para o usuário, e ele deve escolher a partida que deseja apostar.

A cotação é um valor fixo(2, 2.5, 1.5, 3.5, 3 ou 4) o usuário deve escolher esse valor.

Quando for cadastrar uma partida o membro ‘int vencedor’ vai está setado com valor '0', e quando uma partida for finalizada deve ser disparado um processo no qual todas as apostas daquela partida terá o valor do 'vencedor' atualizado para 1.

Não se pode apostar em partidas terminadas. Na aposta o usuário vai escolher uma partida (seu Identificador) que na lista de partidas de uma data específica, mas partidas que já estejam terminadas não entra nessa lista.

Do mesmo modo que para clientes seu sistema deve manter um vetor com a struct

```
typedef struct { float valor,
```

int posicao} ChaveAposta;

O vetor deve ser mantido ordenado pelo sistema.

Isso vai permitir buscas binárias pelo valor da aposta.

4 - Relatórios

Vocês devem criar funções para gerar relatórios (mostrar os dados).

Relatórios Solicitados: Todas as Apostas de um Apostador, apostas em um ano específico, apostas que um Apostador venceu, total ganho e total perdido por um determinado Apostador.

Tais Funções devem alocar os dados solicitados em um vetor para que o usuário possa escolher entre mostrar os dados na tela do monitor ou salvar tais dados em um arquivo do tipo texto (.txt).

5 - Sobre os Vetores contendo a chave e posição no arquivo. (vetor contendo os 'ChaveCliente', por exemplo).

Os vetores devem está ordenados em ordem crescente da chave. Logo, vocês devem implementar uma função de ordenação para esses vetores. Entretanto, quando um novo registro é inserido no vetor previamente ordenado, uma nova ordenação deve ser feita, assim, pode-se 'inserir no vetor' de forma ordenada. Dessa forma, pode-se usar a busca binária para descobrir a posição do elemento no vetor e deslocar todos os elementos para frente e inserir o novo elemento naquela posição.

Esses vetores também ficarão salvos em disco (cada um em um arquivo diferente), entretanto eles serão 'construídos' quando seu programa carregar (começar a rodar) e quando o programa terminar sua execução, os vetores serão salvos novamente no arquivo em disco.

OBS: Esses vetores devem ser alocados dinamicamente, já que não sabemos previamente o tamanho deles.

É interessante manter ainda um arquivo contendo as quantidades de Clientes, Partidas e Apostas. Essas quantidades podem ser salvas em um vetor ou estrutura e seus valores salvos em um arquivo. Isso vai evitar que toda vez que o programa seja aberto tenha que se fazer uma varredura nos arquivos para obter tais quantidades. Ou seja, quando o programa começar sua execução o arquivo com as quantidades é lido, essas quantidades são utilizadas e atualizadas pelo programa e quando o programa fechar as quantidades são armazenadas novamente no arquivo.