Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Organização e Recuperação da Informação
Prof. Tiago A. Almeida <talmeida@ufscar.br>

Trabalho 01

Indexação

Prazo para entrega: 06/10/2025 – 23:59

Atenção

- **E/S:** tanto a entrada quanto a saída de dados devem ser de acordo com os casos de testes abertos;
- **Identificadores de variáveis:** escolha nomes apropriados;
- **Documentação:** inclua comentários e indente corretamente o programa;
- **Erros de compilação:** nota **zero** no trabalho;
- **Tentativa de fraude:** nota **zero na média** para todos os envolvidos. Fraudes, como tentativas de compras de soluções, código gerado por LLMs ou cópias de parte ou de todo código-fonte, de qualquer origem, implicará na reprovação direta na disciplina. Partes do código cujas **ideias** foram desenvolvidas em colaboração com outro(s) aluno(s), geradas por LLMs ou obtidas da Internet, devem ser devidamente documentadas em comentários no referido trecho. Contudo, isso **NÃO** autoriza a cópia de trechos significativos de código, a codificação em conjunto, compra de soluções, uso excessivo de LLMs ou compartilhamento de tela para resolução do trabalho. Em resumo, você pode compartilhar ideias em alto nível, modos de resolver o problema, mas não o código;
- Utilize o código-base e as mensagens pré-definidas (`#define`).

1 Contexto

Você, estudante da Ciência da Computação, estava conversando com seus amigos sobre jogos que jogavam durante a infância em videogames portáteis: as inesquecíveis aventuras em jogos do Mário, os momentos cheios de tensão em Cooking Mafma, e, é claro, as intensas batalhas Pokémon e a notável aventura de tentar capturar todos eles. Porém, para seu choque, nenhum de seus amigos havia jogado os jogos da franquia e, por conta do medo de processos por direitos autorais, todos os simuladores de Pokémon *online* foram encerrados, tornando impossível mostrar aos seus amigos a magia do jogo.

Objetivando mostrar a todos como era jogar quando criança, você decidiu criar uma plataforma com mecânicas diferentes o suficiente do jogo original para evitar um processo por direitos autorais. Nessa plataforma, cada treinador pode se cadastrar e participar de batalhas intensas contra outros membros da comunidade. Além disso, a plataforma permite a conquista de prêmios que atestam o progresso de cada participante, tornando as disputas recompensadoras. Eis que nasce o inigualável **Bolsomon**[1] **Showdown**!

As regras são simples, cada uma das batalhas conta com 12 treinadores que são colocados em uma arena específica e se enfrentam até que apenas um permaneça vitorioso. Cada treinador conta com um time de *bolsomons*, cada um com uma habilidade única que pode ser usada de diversas maneiras. Para participar, a pessoa deve se cadastrar como treinador e adicionar *bolsobolas* (ou aumentar sua quantidade com prêmios). Com as *bolsobolas*, o treinador é capaz de capturar *bolsomons* diferentes e utilizá-los em batalhas, auxiliando-os nos seus intensos duelos.

Para se preparar para o crescimento das atividades do *Bolsomon Showdown*, você decide criar um MVP (*Minimum Viable Product* – Produto Mínimo Viável) capaz de administrar digitalmente os dados das batalhas. Dessa forma, você precisará empregar suas habilidades na linguagem C para criar e manipular de forma eficiente as bases de dados dos treinadores, dos *bolsomons*, das batalhas e dos resultados.

2 Base de dados da aplicação

O sistema será composto por dados dos treinadores, dos *bolsomons*, das batalhas e dos resultados, conforme descrito a seguir.

2.1 Dados dos treinadores

- **id_treinador**: identificador único de um treinador (chave primária), composto por 11 dígitos. Não poderá existir outro valor idêntico na base de dados. Ex: `57956238064`;
- **apelido**: apelido do treinador. Ex: `AshK`;
- **cadastro**: data em que o usuário realizou o cadastro no sistema, no formato `<AAAA><MM><DD><HH><MM>`. Ex: `202509031020`;
- **premio**: última data em que o treinador recebeu uma premiação, no formato `<AAAA><MM><DD><HH><MM>`. Ex: `202509101430`;

[1] *bolsomon*: Monstro de Bolso.

- **bolsobolas**: quantidade de bolsobolas que o treinador possui, no formato `<9999999999>.<99>`. Ex: `0000004605.10`;

2.2 Dados dos *bolsomons*

- **id_bolsomon**: identificador único de cada *bolsomon*, composto por 3 dígitos, no formato `<999>`. Não poderá existir outro valor idêntico na base de dados. Ex: `524`;
- **nome**: nome do *bolsomon*. Ex: `Bolsosaur`;
- **habilidade**: habilidade do *bolsomon*. Ex: `Ataque de vinhas`;
- **preco_bolsobolas**: custo em bolsobolas para capturar o *bolsomon*, no formato `<9999999999>.<99>`. Ex: `00000006000.00`;

2.3 Dados das batalhas

- **id_batalha**: identificador único da batalha, composto por 8 dígitos, no formato `<99999999>`. Não poderá existir outro valor idêntico na base de dados. Ex: `12345678`;
- **inicio**: início da batalha, no formato `<AAAA><MM><DD><HH><MM>`. Ex: `202509031020`
- **duracao**: tempo de duração da batalha, no formato `<HH><MM><SS>`. Ex: `012044`;
- **arena**: identificador da arena da batalha com 4 dígitos, no formato `<9999>`. Ex: `0136`;

2.4 Dados de estatísticas das batalhas

- **id_treinador**: ID do treinador;
- **id_batalha**: ID da batalha;
- **id_bolsomon**: ID do *bolsomon* que o treinador utilizou na batalha;
- **foi_maior_duracao**: valor 1 se o treinador foi o que permaneceu por mais tempo na batalha, 0 caso contrário;
- **foi_mais_derrotas**: valor 1 se o treinador foi o que derrotou o maior número de *bolsomons* na batalha, 0 caso contrário;
- **foi_mais_dano**: valor 1 se o treinador foi o que causou a maior quantidade de dano na batalha, 0 caso contrário;

2.5 Dados de posse de *bolsomons*

Esta tabela associativa relaciona os treinadores aos *bolsomons* que eles possuem. Cada linha representa um único *bolsomon* no time de um treinador.

- **id_treinador**: ID do treinador (chave estrangeira).

- **id_bolsomon**: ID do *bolsomon* possuído (chave estrangeira).

Garantidamente, nenhum campo de texto receberá caracteres acentuados.

2.6 Criação das bases de dados em SQL

Cada base de dados corresponde a um arquivo distinto. Elas poderiam ser criadas em um banco de dados relacional usando os seguintes comandos SQL:

```
CREATE TABLE treinadores (
  id_treinador  varchar(11) NOT NULL PRIMARY KEY,
  apelido       text NOT NULL,
  cadastro      varchar(12) NOT NULL,
  premio        varchar(12) NOT NULL DEFAULT '000000000000',
  bolsobolas    numeric(12, 2) NOT NULL DEFAULT 0
);
CREATE TABLE bolsomons (
  id_bolsomon       varchar(3) NOT NULL PRIMARY KEY,
  nome              text NOT NULL,
  habilidade        text NOT NULL,
  preco_bolsobolas  numeric(12,2) NOT NULL
);
CREATE TABLE treinador_possui_bolsomon (
  id_treinador  varchar(11) NOT NULL,
  id_bolsomon   varchar(3) NOT NULL,
  PRIMARY KEY   (id_treinador, id_bolsomon)
);
CREATE TABLE batalhas (
  id_batalha  varchar(8) NOT NULL PRIMARY KEY,
  inicio      varchar(12) NOT NULL,
  duracao     varchar(6) NOT NULL,
  arena       varchar(4) NOT NULL
);
CREATE TABLE resultados (
  id_treinador       varchar(11) NOT NULL,
  id_batalha         varchar(8) NOT NULL,
  id_bolsomon        varchar(3) NOT NULL,
  foi_maior_duracao  numeric(1, 0) NOT NULL,
  foi_mais_derrotas  numeric(1, 0) NOT NULL,
  foi_mais_dano      numeric(1, 0) NOT NULL
);
```

3 Operações suportadas pelo programa

Os dados devem ser manipulados através do console/terminal (modo texto) usando uma sintaxe similar à SQL, sendo que as operações a seguir devem ser fornecidas.

3.1 Cadastro de treinadores

```
INSERT INTO treinadores VALUES ('<id_treinador>', '<apelido>');
```

Para criar uma nova conta de treinador, seu programa deve ler os campos `id_treinador` e `apelido`. Inicialmente, a conta será criada sem bolsobolas (`00000000000.00`). O campo `cadastro` receberá a data em que o cadastro foi realizado. A função deve falhar caso haja a tentativa de inserir um treinador com um `id_treinador` já cadastrado. Neste caso, deverá ser apresentada a mensagem de erro padrão `ERRO_PK_REPETIDA`. Caso a operação se concretize com sucesso, exibir a mensagem padrão `SUCESSO`.

3.2 Remoção de treinadores

```
DELETE FROM treinadores WHERE id_treinador = '<id_treinador>';
```

O usuário deverá ser capaz de remover uma conta dado um ID de um treinador. Caso a conta não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. A remoção na base de dados deverá ser feita por meio de um marcador, conforme descrito na Seção 6. Se a operação for realizada, exibir a mensagem padrão `SUCESSO`.

3.3 Adicionar bolsobolas na conta

```
UPDATE treinadores SET bolsobolas = bolsobolas + '<valor>' WHERE id_treinador =
'<id_treinador>';
```

O usuário deverá ser capaz de adicionar bolsobolas na conta de um treinador dado seu ID e o valor desejado. Caso o treinador não esteja cadastrado no sistema, o programa deverá imprimir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o valor que esteja sendo adicionado seja menor ou igual a zero, o programa deve imprimir a mensagem `ERRO_VALOR_INVALIDO`. Se não houver nenhum desses problemas, as bolsobolas deverão ser atualizadas, seguido da impressão da mensagem padrão de `SUCESSO`.

3.4 Capturar um *bolsomon*

```
INSERT INTO treinador_possui_bolsomon VALUES ('<id_treinador>', '<id_bolsomon>');
```

O usuário poderá capturar um *bolsomon* para um treinador dado o ID do treinador e o ID do *bolsomon* desejado, caso possua bolsobolas para capturá-lo. Neste caso, o valor será descontado da conta do treinador. Caso o treinador não possua bolsobolas suficientes, a mensagem padrão `ERRO_SALDO_NAO_SUFICIENTE` deverá ser impressa. Caso o treinador ou o *bolsomon* não exista, o programa deverá imprimir `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o treinador já possua o *bolsomon* (ou seja, a combinação de IDs já existe na tabela de posse), seu programa deverá imprimir

ERRO_BOLSOMON_REPETIDO. Caso não haja nenhum erro, o programa deve inserir uma nova linha na tabela de posse, atualizando todos os índices e arquivos necessários e, então, imprimir a mensagem padrão SUCESSO.

3.5 Cadastro de *bolsomons*

```
INSERT INTO bolsomons VALUES ('<nome>', '<habilidade>', '<preco_bolsobolas>');
```

Para um *bolsomon* ser adicionado no banco de dados, seu programa deverá ler os campos nome, habilidade e preco_bolsobolas. O campo id_bolsomon segue a ordem de cadastro dos *bolsomons*, sendo incrementado a cada novo *bolsomon* registrado. Caso a operação se concretize, exiba a mensagem padrão SUCESSO.

3.6 Iniciar batalha

```
INSERT INTO batalhas VALUES ('<inicio>', '<duracao>', '<arena>');
```

Para iniciar uma batalha, seu programa deve ler os campos inicio, duracao e arena para criar um novo registro na tabela batalhas. O campo id_batalha será incremental. Em seguida, os dados de cada participante (informados através de treinadores, bolsomons_treinadores, duracoes_treinadores, eliminacoes e danos_causados) serão processados para criar os registros correspondentes na tabela resultados, preenchendo as conquistas (foi_maior_duracao, etc.) conforme a performance de cada um.

No caso do ID de algum treinador não existir, ou ter sido deletado, exibir a mensagem ERRO_REGISTRO_NAO_ENCONTRADO. Caso o treinador não possua o *bolsomon* atribuído a ele na batalha, exibir ERRO_TREINADOR_BOLSOMON. Em qualquer uma dessas situações, a operação de inserção deve ser interrompida. Se a operação for concretizada com sucesso, exibir a mensagem padrão SUCESSO.

3.7 Recompensar campeão

```
RECOMPENSAR_CAMPEAO('<data_inicio>', '<data_fim>', '<valor_premio>');
```

Ao executar a função, o treinador com o melhor desempenho entre data_inicio e data_fim deverá ser recompensado com o valor_premio em bolsobolas. O campo premio, em treinadores, deve armazenar o final do período da premiação mais recente concedida ao treinador. Quando a operação se concretizar com sucesso, exibir a mensagem padrão CONCEDER_PREMIO.

Note que registros de treinadores podem ser removidos nesse sistema. Na situação em que o treinador com melhor desempenho no período tiver sido excluído do sistema, exibir a mensagem padrão ERRO_TREINADOR_REMOVIDO e encontrar o próximo treinador com melhor desempenho para conceder o prêmio.

Para descobrir qual treinador obteve o melhor desempenho no período, será calculado um **placar de conquistas** para cada tipo de conquista, sendo esses o número de vezes que aquele treinador conseguiu cada conquista naquele período.

As seguintes regras de desempate serão aplicadas em ordem:

(a) O treinador com o maior **placar de tempo** é aquele que obteve o melhor desempenho.

(b) Em caso de empate no placar, o treinador com maior **placar de derrotas** será recompensado.

(c) Caso o empate não tenha sido resolvido, o treinador com o maior **placar de dano** ganhará o prêmio.

(d) Se o empate persistir, a escolha daquele com melhor desempenho será decidida pelo menor valor `id_treinador`.

3.8 Busca

As seguintes operações de busca por treinadores, *bolsomons* e batalhas deverão ser implementadas. *Em todas elas, será necessário utilizar a busca binária e mostrar o caminho percorrido nos índices da seguinte maneira*:

```
Registros percorridos:   3 2 0 1
```

No exemplo acima, os números representam o RRN dos registros que foram percorridos durante a busca até encontrar o registro de interesse ou esgotar as possibilidades.

ATENÇÃO: em todos os cenários de busca binária, caso o número de elementos seja par (p.ex, 10 elementos), então há 2 (duas) possibilidades para a posição da mediana dos elementos (p.ex., 5^a ou 6^a posição se o total fosse 10). Neste caso, **sempre** escolha a posição mais à direita (p.ex., a posição 6 caso o total for 10).

3.8.1 Treinador

O usuário deverá poder buscar contas de treinadores pelos seguintes atributos:

(a) ID do treinador:

```
SELECT * FROM treinadores WHERE id_treinador = '<id_treinador>';
```

Solicitar ao usuário o ID de cadastro do treinador. Caso a conta não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o cadastro exista, todos os seus dados deverão ser impressos na tela de forma formatada.

3.8.2 *Bolsomon*

O usuário deverá ser capaz de buscar *bolsomons* pelos seguintes atributos:

(a) ID do *bolsomon*:

```
SELECT * FROM bolsomons WHERE id_bolsomon = '<id_bolsomon>';
```

Solicitar ao usuário o ID do *bolsomon*. Caso não exista no banco de dados, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso contrário, todos os dados do *bolsomon* deverão ser impressos na tela de forma formatada.

3.8.3 Batalha

O usuário deverá ser capaz de buscar batalhas pelos seguintes atributos:

(a) ID da batalha:

```
SELECT * FROM batalhas WHERE id_batalha = '<id_batalha>';
```

Solicitar ao usuário o ID da batalha. Caso não exista no banco de dados, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso contrário, todos os dados da batalha deverão ser impressos na tela de forma formatada.

3.9 Listagem

As seguintes operações de listagem deverão ser implementadas.

3.9.1 Treinadores

(a) Pelos IDs dos treinadores:

```
SELECT * FROM treinadores ORDER BY id_treinador ASC;
```

Exibe todos os treinadores ordenados de forma crescente pelo ID. Caso nenhum registro seja retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

(b) Por nome de *bolsomon*:

```
SELECT T.*
FROM treinadores T
INNER JOIN treinador_possui_bolsomon TPB ON T.id_treinador =
TPB.id_treinador
INNER JOIN bolsomons B ON TPB.id_bolsomon = B.id_bolsomon
WHERE B.nome = '<nome_bolsomon>'
ORDER BY T.id_treinador;
```

Exibe todos os treinadores que possuem determinado *bolsomon*, em ordem crescente de ID. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

ATENÇÃO: antes da listagem dos treinadores, o seu programa deverá imprimir os registros do índice da lista invertida (*i.e.*, `treinador_bolsomons_primario_idx`, ver detalhes na Seção 4.2) que foram percorridos na listagem (obs: a busca no índice secundário, contendo os nomes dos *bolsomons*, deverá ser feita por uma busca binária).

Exemplo:

```
Registros percorridos:   3 5 6
```

No exemplo acima, os valores representam o número do índice dos registros que foram percorridos durante a busca até encontrar o registro de interesse ou esgotar as possibilidades no índice secundário.

3.9.2 *Bolsomons* para capturar

(a) De acordo com as bolsobolas do treinador:

```
SELECT * FROM bolsomons WHERE preco_bolsobolas <= ('SELECT bolsobolas FROM
treinador WHERE id_treinador = <id_treinador> ');
```

Seu programa deve ler o ID de um treinador e, em seguida, exibir todos os *bolsomons* que o treinador pode capturar, de acordo com suas bolsobolas. Na situação em que o treinador foi removido, imprimir a mensagem `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

3.9.3 Batalha

(a) Por data de batalha:

```
SELECT * FROM batalhas WHERE inicio BETWEEN '<data_inicio>' AND '<data_fim>'
ORDER BY inicio ASC;
```

Exibe todas as batalhas realizadas em um determinado intervalo (data entre `<data_inicio>` e `<data_fim>`), em ordem cronológica. Ambas as datas estarão no formato `<AAAAMMDDHHMM>`. Para cada registro encontrado na listagem, deverá ser impresso o caminho percorrido. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

ATENÇÃO: antes de imprimir a lista de batalhas realizadas no período, primeiro é necessário imprimir o caminho percorrido durante a busca binária para encontrar o registro cujo `<inicio>` seja igual à `<data_inicio>` informada pelo usuário ou data posterior mais próxima.

3.10 Liberar espaço

```
VACUUM treinadores;
```

O `ARQUIVO_TREINADORES` deverá ser reorganizado com a remoção física de todos os registros marcados como excluídos e os índices deverão ser atualizados. A ordem dos registros no arquivo "limpo" não deverá ser diferente do arquivo "sujo". Se a operação se concretizar, exibir a mensagem padrão `SUCESSO`.

3.11 Imprimir arquivos de dados

O sistema deverá imprimir os arquivos de dados da seguinte maneira:

(a) Dados dos treinadores:

```
echo file ARQUIVO_TREINADORES
```

Imprime o arquivo de dados de treinadores. Caso estiver vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`;

(b) Dados dos *bolsomons*:

```
echo file ARQUIVO_BOLSOMONS
```

Imprime o arquivo de dados de *bolsomons*. Caso estiver vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

(c) Dados da relação treinador-*bolsomon*:

```
echo file ARQUIVO_TREINADOR_POSSUI_BOLSOMON
```

Imprime o arquivo de dados da relação treinador-*bolsomon*. Caso estiver vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

(d) Dados das batalhas:

```
echo file ARQUIVO_BATALHAS
```

Imprime o arquivo de dados de batalhas. Caso o arquivo estiver vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

(e) Dados de resultados:

```
echo file ARQUIVO_RESULTADOS
```

Imprime o arquivo de resultados. Caso o arquivo estiver vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

3.12 Imprimir índices primários

O sistema deverá imprimir os índices primários da seguinte maneira:

(a) Índice de treinadores com `id_treinador` e `rrn`:

```
echo index treinadores_idx
```

Imprime as *structs* de índice primário de treinadores. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(b) Índice de *bolsomons* com `id_bolsomon` e `rrn`:

```
echo index bolsomons_idx
```

Imprime as *structs* de índice primário de *bolsomons*. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(c) Índice de batalhas com `id_batalha` e `rrn`:

```
echo index batalhas_idx
```

Imprime as *structs* de índice primário de batalhas. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(d) Índice de resultados com `id_treinador`, `id_batalha` e `rrn`:

```
echo index resultados_idx
```

Imprime as *structs* de índice primário de resultados. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(e) Índice da relação treinador-*bolsomon* com `id_treinador`, `id_bolsomon` e `rrn`:

```
echo index treinador_possui_bolsomon_idx
```

Imprime as *structs* de índice primário da relação treinador-*bolsomon*q. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

3.13 Imprimir índices secundários

O sistema deverá imprimir os índices secundários da seguinte maneira:

(a) Índice de preço com `preco_bolsobolas` e `id_bolsomon`:

```
echo index preco_bolsomon_idx
```

Imprime as *structs* de índice secundário de *bolsomons*. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(b) Índice de data das batalhas com `inicio` e `id_batalha`:

```
echo index data_idx
```

Imprime as *structs* de índice secundário de datas das batalhas. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(c) Índice de treinador *bolsomons* secundário:

```
echo index treinador_bolsomons_secundario_idx
```

Imprime as *structs* de índice secundário com os nomes dos *bolsomons* (`treinador_bolsomons_secundario_idx`) e o número do índice do primeiro treinador que possui esse *bolsomon*. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(d) Índice de treinador *bolsomons* primário (obs: no escopo deste trabalho, este índice também é secundário):

```
echo index treinador_bolsomons_primario_idx
```

Imprime as *structs* de índice secundário com o ID de um treinador que possui o *bolsomon* (`treinador_bolsomons_primario_idx`) e número do índice do próximo treinador que tenha esse *bolsomon*. Caso o índice estiver vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

3.14 Finalizar

Libera a memória e encerra a execução do programa.

```
q
```

4 Criação dos índices

Para que as buscas e as listagens ordenadas dos dados sejam otimizadas, é necessário criar e manter índices em memória (que serão liberados ao término do programa). *Todas as chaves devem ser armazenadas na forma canônica, em caixa alta.*

Pelo menos os seguintes índices deverão ser criados:

4.1 Índices primários

- `treinadores_idx`: índice primário que contém o ID do treinador (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pelo ID do treinador (`id_treinador`);
- `bolsomons_idx`: índice primário que contém o ID do *bolsomon* (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pelo ID do *bolsomon* (`id_bolsomon`);
- `batalhas_idx`: índice primário que contém o ID da batalha (chave primária) e o RRN do respectivo registro no arquivo de batalhas, ordenado pelo ID da batalha (`id_batalha`);
- `resultados_idx`: índice primário que consiste no ID de um treinador, ID de uma batalha e o RRN do respectivo registro no arquivo de resultados, ordenado pelo ID do treinador (`id_treinador`) e ID da batalha (`id_batalha`).
- `treinador_possui_bolsomon_idx`: índice primário que consiste no ID de um treinador, o ID do *bolsomon* associado a ele e o RRN do respectivo registro no arquivo de treinadores que possuem *bolsomons*, ordenado pelo ID do treinador (`id_treinador`) e ID do *bolsomon* (`id_bolsomon`).

4.2 Índices secundários

- `preco_bolsomon_idx`: índice secundário que contém os *bolsomons* ordenados por preço e a chave primária (`id_bolsomon`) do *bolsomon* específico.
- `data_idx`: índice secundário que contém as *batalhas* ordenadas pela data e horário de início e a chave primária (`id_batalha`) da *batalha* específica.
- `treinador_bolsomons_idx`: índice secundário do tipo *lista invertida*. Será necessário manter dois índices (`treinador_bolsomons_primario_idx` e `treinador_bolsomons_secundario_idx`), sendo que o primário possui os IDs de treinadores (`id_treinador`) que possuem certo *bolsomon* e o apontador para o próximo treinador que possui o mesmo *bolsomon* nesse mesmo índice primário. Se não houver um próximo treinador, esse apontador deve possuir o valor `-1`. No índice secundário estão os nomes dos *bolsomons*, assim como a referência do primeiro treinador que possui aquele *bolsomon* no índice primário.

  Para simplificar o entendimento, considere o seguinte exemplo:

Treinador com *Bolsomons* Primário

ID do Treinador	Próximo Registro
0	4
4	5
4	7
4	-1
3	-1
7	6
5	-1
5	-1
...	...

Treinador com *Bolsomons* Secundário

Nome do *Bolsomon*	Registro
PATU	0
JACARÉ DE OSSO	1
BALBOA	3
ILSON	2
...	...

No exemplo acima, a tabela de Treinador com *Bolsomons* Secundário possui o nome do *bolsomon* na primeira coluna, assim como o RRN do primeiro treinador que possui aquele *bolsomon*, que foi inserido na tabela de Treinador com *Bolsomons* Primário. Na tabela primária, tem-se na primeira coluna o ID dos treinadores que possuem cada *bolsomon*. Note que o treinador com ID = 4 aparece três vezes no exemplo, o que significa que ele possui três *bolsomons* diferentes. Na segunda coluna da tabela primária, temos o RRN para o próximo treinador que possui o mesmo *bolsomon* na própria tabela de Treinador com *Bolsomons* Primário, sendo que, RRN = -1, significa que aquele treinador é o último que possui tal bolsomon. Vale destacar que o índice primário **não** precisa estar organizado, pois cada registro já possui uma referência direta para o próximo (assim como em uma lista encadeada).

Deverá ser desenvolvida uma rotina para a criação de cada índice. Eles serão sempre criados e manipulados em memória principal na inicialização e liberados ao término do programa. Note que o ideal é que os índices primários sejam criados primeiro, depois os secundários.

Após a criação de cada arquivo de índice, deverá ser impressa na tela a frase padrão `INDICE_CRIADO`.

5 Arquivos de dados

Como este trabalho será corrigido automaticamente por um juiz online que não aceita funções que manipulam arquivos, os registros serão armazenados e manipulados em *strings* que irão simular os arquivos abertos. Para isso, você deverá utilizar as variáveis globais `ARQUIVO_TREINADORES`, `ARQUIVO_BOLSOMONS`, `ARQUIVO_TREINADOR_POSSUI_BOLSOMON`, `ARQUIVO_BATALHAS` e `ARQUIVO_RESULTADOS`, e as funções de leitura e escrita em *strings*, como `sprintf` e `sscanf`, para simular as operações de leitura e escrita em arquivo. Os arquivos de dados devem ser no formato ASCII (arquivo texto).

Na implementação em C, os *buffers* de *string* (`char[]`) terão um byte adicional para o caractere nulo terminador (`\0`), conforme definido no arquivo de cabeçalho por meio dos `#define`. Por isso, `TAM_ID_TREINADOR` será 11 (quantidade de caracteres no `ID`) + 1 (para o `\0`) = 12.

`ARQUIVO_TREINADORES`: deverá ser organizado em registros de tamanho fixo de 96 *bytes* (96 caracteres).

O campo `apelido` (tamanho máximo de 43 *bytes*) deve ser de tamanho variável.

Os demais campos são de tamanho fixo e possuem as seguintes especificações: `id_treinador` (11 *bytes*), `cadastro` (12 *bytes*), `premio` (12 *bytes*), `bolsobolas` (13 *bytes*), totalizando 48 *bytes* de tamanho fixo. Portanto, os campos de tamanho fixo de um registro ocuparão 48 *bytes*. Os campos devem ser separados pelo caractere delimitador '`;`' (ponto e vírgula), e cada registro terá 5 delimitadores (um para cada campo). Caso o registro tenha menos de 96 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 48 *bytes* fixos + 5 *bytes* de delimitadores, então os campos variáveis devem ocupar no máximo 43 *bytes*, para que o registro não exceda os 96 *bytes*.

Exemplo de arquivo de dados de treinadores:

```
57956238064;AshK;202509031020;202509101430;0000004605.10;########################
##############12345678901;Misty;202509041200;202509061010;0000000020.50;#########
############################98765432109;Brock;202509051530;000000000000;0000000
005.00;#####################################11122233344;Gary;202509021115;202509
051230;0000001000.00;#####################################
```

`ARQUIVO_BOLSOMONS`: o arquivo de *bolsomons* deverá ser organizado em registros de tamanho fixo de `100` *bytes* (*i.e.*, `100` caracteres).

Os campos `nome` (máximo de 20 *bytes*) e `habilidade` (máximo de 60 *bytes*) devem ser de tamanhos variáveis. Os demais campos são de tamanho fixo e possuem as seguintes especificações: `id_bolsomon` (3 *bytes*) e `preco_bolsobolas` (13 *bytes*), totalizando 16 *bytes* de tamanho fixo. Assim como no registro de treinadores, os campos devem ser separados pelo delimitador '`;`', cada registro terá 4 delimitadores para os campos e, caso o registro tenha menos que `100` *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 16 *bytes* de tamanho fixo + 4 *bytes* para os delimitadores, os campos variáveis devem ocupar no máximo `80` *bytes* para que não se exceda o tamanho do registro.

Exemplo de arquivo de dados de *bolsomons*:

```
001;Bolsosaur;Ataquedevinhas;0000000600.00;#####################################
###################002;Charbolso;Chamaeterna;00000007500.00;####################
#####################################003;Bolsoturtle;Jatodeagua;00000005000.00;#
#########################################################004;Bolsochu;Choquedotro
vao;00000008000.00;###########################################################
```

`ARQUIVO_TREINADOR_POSSUI_BOLSOMON`: deverá ser organizado em registros de tamanho fixo de 14 *bytes* (14 caracteres). Todos os campos são de tamanho fixo: `id_treinador` (11 *bytes*) e `id_bolsomon` (3 *bytes*).

Exemplo de arquivo de dados de treinador possui *bolsomon*:

```
57956238064001579562380640041234567890100398765432109002111222333440011
11222333440021112223334400 4
```

`ARQUIVO_BATALHAS`: o arquivo de batalhas deverá ser organizado em registros de tamanho fixo de 30 *bytes* (30 caracteres). Todos os campos são de tamanho fixo e possuem as seguintes especificações: `id_batalha` (8 *bytes*), `inicio` (12 *bytes*), `duracao` (6 *bytes*) e `arena` (4 *bytes*).

Exemplo de arquivo de dados de batalhas:

```
00000001202509031020012044013600000002202509041200010530020100000003 2025
09051000013015045000000004202509061100004510100000000005202509061230021 0
05235900000006202509070900005520050100000007202509071030010000121200000
00820250908140000304530000000000920250908151502151045000000010202509091
00000400050500000001120250909113001050060600000001220250910130000253070
70000000132025091014450140008080000000142025091109000050109090000000152
025091110150110051010
```

`ARQUIVO_RESULTADOS`: o arquivo de resultados deverá ser organizado em registros de tamanho fixo de 25 *bytes* (25 caracteres). Todos os campos são de tamanho fixo e possuem as seguintes especificações: `id_treinador` (11 *bytes*), `id_batalha` (8 *bytes*), `id_bolsomon` (3 *bytes*), `foi_maior_duracao` (1 *byte*), `foi_mais_derrotas` (1 *byte*) e `foi_mais_dano` (1 *byte*).

Exemplo de arquivo de dados de resultados:

```
57956238064000000010011001234567890100000001003000987654321090000000
10020001112223334400000001004000222333444550000000100500033344445
55660000000100600044455566677000000010070005556667778800000001008 0
00666777888990000000100901177788899900000000010100008889990001100
00000101100099900011122000000010120001234567890100000002013100 98
76543210900000002014000111222333440000000201500022233344455000000 02
01600133344455566000000020170004445556667700000002018000555666777
88000000020190106667778889900000002020000777888999000000000202100
08889990001100000002022000999000111220000000202300000011122233000 0
0002024000
```

O exemplo acima exibe os resultados de duas batalhas, com 12 participantes cada. Note que, em cada batalha, a concessão de `foi_maior_duracao` é exclusiva a um treinador; da mesma forma, `foi_mais_derrotas` é exclusiva a um treinador, e `foi_mais_dano` também é exclusiva a um treinador. Ou seja, dentro de uma batalha, não pode haver dois treinadores com esses atributos igual a 1, visto que esses atributos sinalizam os maiores valores de suas respectivas batalhas.

6 Instruções para as operações com os registros

- **Inserção**: cada treinador, *bolsomon*, batalha e resultado deverá ser inserido no final de seus respectivos arquivos de dados, e atualizados os índices.
- **Remoção**: o registro de um dado treinador deverá ser localizado acessando o índice primário (`id_treinador`). A remoção deverá colocar o marcador `*|` nas duas primeiras posições do registro removido. O espaço do registro removido não deverá ser reutilizado para novas inserções. Observe que o registro deverá continuar ocupando exatamente `TAM_REGISTRO_TREINADOR` *bytes*. Além disso, no índice primário, o `RRN` correspondente ao registro removido deverá ser substituído por `-1`.
- **Atualização**: existem dois campos alteráveis: (*i*) as bolsobolas do treinador e (*ii*) data da última premiação recebida pelo treinador. Eles possuem tamanho fixo e pré-determinado. A atualização deve ser feita diretamente no registro, exatamente na mesma posição em que estiverem (em hipótese alguma o registro deverá ser removido e em seguida inserido).

7 Inicialização do programa

Para que o programa inicie corretamente, deve-se realizar o seguinte procedimento:

1. Inserir o comando `SET ARQUIVO_TREINADORES TO '<DADOS DE TREINADORES>';` caso queira inicializar o programa com um arquivo de treinadores já preenchido;
2. Inserir o comando `SET ARQUIVO_BOLSOMONS TO '<DADOS DE BOLSOMONS>';` caso queira inicializar o programa com um arquivo de *bolsomons* já preenchido;
3. Inserir o comando `SET ARQUIVO_BATALHAS TO '<DADOS DE BATALHAS>';` caso queira inicializar o programa com um arquivo de batalhas já preenchido;
4. Inserir o comando `SET ARQUIVO_RESULTADOS TO '<DADOS DE RESULTADOS>';` caso queira inicializar o programa com um arquivo de resultados já preenchido;
5. Inicializar as estruturas de dados dos índices.

8 Implementação

Implemente suas funções utilizando o código-base fornecido. **Não é permitido modificar os trechos de código pronto ou as estruturas já definidas**. Ao imprimir um registro, utilize as funções `exibir_treinador(int rrn)`, `exibir_bolsomon(int rrn)`, `exibir_batalha(int rrn)` ou `exibir_resultado(int rrn)`.

Implemente as rotinas abaixo com, obrigatoriamente, as seguintes funcionalidades:

- Estruturas de dados adequadas para armazenar os índices na memória principal;
- Verificar se os arquivos de dados existem;
- Criar os índices primários: deve refazer os índices primários a partir dos arquivos de dados;
- Criar os índices secundários: deve refazer os índices secundários a partir dos arquivos de dados;
- Inserir um registro: modifica os arquivos de dados e os índices na memória principal;
- Buscar por registro: busca pela chave primária ou por uma das chaves secundárias;
- Alterar um registro: modifica o campo do registro diretamente no arquivo de dados;
- Remover um registro: modifica o arquivo de dados e o índice primário na memória principal;
- Listar registros: listar todos os registros ordenados pela chave primária ou por uma das chaves secundárias;
- Liberar espaço: organizar o arquivo de dados e refazer os índices.

Lembre-se de que, sempre que possível, é **obrigatório o uso da busca binária**, com o arredondamento para **cima** para buscas feitas em índices tanto primários quanto secundários.

9 Dicas

- **ATENÇÃO:** É fortemente recomendado que você inicie o trabalho o mais breve possível, pois você PRECISARÁ dedicar várias horas para conseguir concluí-lo.

- Ao ler uma entrada, tome cuidado com caracteres de quebra de linha (`\n`) não capturados;
- Você nunca deve perder a referência do começo do arquivo, então não é recomendável percorrer a *string* diretamente pelo ponteiro `ARQUIVO`. Um comando equivalente a `fseek(f, 256, SEEK_SET)` é `char *p = ARQUIVO + 256`;
- Diferentemente do `fscanf`, o `sscanf` não movimenta automaticamente o ponteiro após a leitura;
- O `sprintf` adiciona automaticamente o caractere `\0` no final da *string* escrita. Em alguns casos você precisará sobrescrever a posição manualmente. Você também pode utilizar o comando `strncpy` para escrever em *strings*. Esse comando, diferentemente do `sprintf`, não adiciona o caractere nulo no final;
- A função `strtok` permite navegar nas *substrings* de uma certa *string* dado o(s) delimitador(es). Porém, tenha em mente que ela deve ser usada em uma cópia da *string* original, pois ela modifica o primeiro argumento;
- É recomendado olhar as funções `qsort` e `bsearch` da biblioteca C. Caso se baseie nelas para a sua implementação, leia suas documentações;
- Para o funcionamento ideal do seu programa, é necessário utilizar a busca binária.

"Diante de uma larga frente de batalha, procure o ponto mais fraco
e, ali, ataque com a sua maior força"
— Sun Tzu