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: 27/11/2023 – 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 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) devem ser devidamente documentadas em comentários no referido trecho. Contudo, isso **NÃO** autoriza a cópia de trechos de código, a codificação em conjunto, compra de soluções, 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

Durante o período de ensino a distância, alunos das universidades brasileiras diminuíram o contato entre si e perderam experiências memoráveis que a graduação poderia proporcionar. Nesse contexto, um grupo de universitários reuniu recursos e criou uma atividade de extensão cujo objetivo seria fomentar a conexão entre os estudantes. Nela, alunos de todos os *campi* da universidade se reuniriam em pistas e disputariam corridas de *kart*, com direito a prêmios, socialização e diversão. Essa atividade foi carinhosamente chamada de **UFSKart**.

Com a rápida adesão da comunidade à atividade de extensão, as regras se tornaram claras: para participar, um aluno deve se cadastrar como corredor e adicionar saldo a sua conta. Com o saldo, o corredor é capaz de "comprar" modelos de veículos diferentes e pilotá-los em pistas, competindo contra outros corredores nas corridas.

Contudo, com o crescimento dos eventos, percebeu-se a necessidade de se armazenar os dados de maneira conveniente e de fácil manipulação. Dessa forma, os organizadores da UFSKart chamaram você, exímio programador, para criar um MVP, produto mínimo viável, capaz de administrar digitalmente os dados das corridas. Para isso, você precisará empregar suas habilidades na linguagem C para criar e manipular de forma eficiente as bases de dados dos corredores, dos veículos, das pistas e das corridas para essa atividade.

2 Base de dados da aplicação

O sistema será composto por dados dos corredores, dos veículos, das pistas e das corridas, conforme descrito a seguir.

2.1 Dados dos corredores

- **id_corredor**: identificador único de um corredor (chave primária), composto pelo seu CPF (11 dígitos). Não poderá existir outro valor idêntico na base de dados. Ex: `57956238064`;
- **nome**: nome do corredor. Ex: `Arnaldo Turbinaldo`;
- **apelido**: apelido do corredor. Ex: `Turbi-Arnaldo`;
- **cadastro**: data em que o usuário realizou o cadastro no sistema, no formato `<AAAA><MM><DD><HH><MM>`. Ex: `202201021020`;
- **saldo**: saldo que o usuário possui na sua conta, no formato `<9999999999>.<99>`. Ex: `0000004605.10`;
- **veiculos**: até três modelos de veículos adquiridos pelo corredor, separados por '|'. Ex: `CCG-Turbo|Vivenciomovel|Tetrakyklo|`;

2.2 Dados dos veículos

- **id_veiculo**: identificador único de cada veículo, composto por 7 dígitos, no formato `<9999999>`. Não poderá existir outro valor idêntico na base de dados. Ex: `5247812`;

- **marca**: marca do veículo. Ex: `Pedrolet`;
- **modelo**: modelo do veículo. Ex: `PACmobile`;
- **poder**: poder que o veículo possui. Ex: `Fazer o oponente chorar`;
- **velocidade**: número que indica o nível de velocidade do veículo, no formato `<9999>`. Ex: `0007`;
- **aceleracao**: número que indica o nível de aceleração do veículo, no formato `<9999>`. Ex: `0010`;
- **peso**: número que indica o nível do peso do veículo, no formato `<9999>`. Ex: `0005`;
- **preco**: valor do veículo no formato `<99999999999>.<99>`. Ex: `00000006000.00`;

2.3 Dados das pistas

- **id_pista**: identificador único da pista, composto por 8 dígitos, no formato `<99999999>`. Não poderá existir outro valor idêntico na base de dados. Ex: `12345678`;
- **nome**: nome da pista. Ex: `Cachoeira Misteriosa`
- **dificuldade**: nível de dificuldade da pista, no formato `<9999>`. Ex: `0002`;
- **distancia**: distância da pista, no formato `<9999>`. Ex: `0250`;
- **recorde**: tempo recorde da pista, no formato `<9999>`. Ex: `0136`;

2.4 Dados das corridas

- **id_pista**: ID da pista onde será realizada a corrida;
- **ocorrencia**: data da realização da corrida, no formato `<AAAA><MM><DD><HH><MM>`;
- **id_corredores**: IDs dos seis corredores participantes, organizado por ordem de chegada, no formato `<ID_CORREDOR_1> <ID_CORREDOR_2> <ID_CORREDOR_3> <ID_CORREDOR_4> <ID_CORREDOR_5> <ID_CORREDOR_6>`. Ex: `375678765425856744654246567876542005678765567810567125678765 42`
- **id_veiculos**: IDs dos veículos utilizados pelos corredores, organizados por ordem de chegada, no formato `<ID_VEICULO_1> <ID_VEICULO_2> <ID_VEICULO_3> <ID_VEICULO_4> <ID_VEICULO_5> <ID_VEICULO_6>`. Ex: `00000000000001000000200000030000000400000 05`.

Garantidamente, nenhum campo de texto receberá caracteres acentuados.

2.5 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 corredores (
  id_corredor  varchar(11) NOT NULL PRIMARY KEY,
  nome         text NOT NULL,
  apelido      text NOT NULL,
  cadastro     varchar(12) NOT NULL,
  saldo        numeric(12, 2) DEFAULT 0,
  veiculos     text[3] DEFAULT '{}'
);
CREATE TABLE veiculos (
  id_veiculo  varchar(7) NOT NULL PRIMARY KEY,
  marca       text NOT NULL,
  modelo      text NOT NULL,
  poder       text NOT NULL,
  velocidade  numeric(4, 0) NOT NULL,
  aceleracao  numeric(4, 0) NOT NULL,
  peso        numeric(4, 0) NOT NULL,
  preco       numeric(12, 2) NOT NULL
);
CREATE TABLE pistas (
  id_pista     varchar(8) NOT NULL PRIMARY KEY,
  nome         text NOT NULL,
  dificuldade  numeric(4, 0) NOT NULL DEFAULT 1,
  distancia    numeric(4, 0) NOT NULL,
  recorde      numeric(4, 0) NOT NULL
);
CREATE TABLE corridas (
  id_pista       varchar(8) NOT NULL,
  ocorrencia     varchar(12) NOT NULL,
  id_corredores  varchar(66) NOT NULL,
  id_veiculos    varchar(42) 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 corredores

```
INSERT INTO corredores VALUES ('<id_corredor>', '<nome>', '<apelido>', '<cadas-
tro>');
```

Para criar uma nova conta de corredor, seu programa deve ler os campos `cpf`, `nome`, `apelido` e `cadastro`. Inicialmente, a conta será criada sem saldo (`00000000000.00`). Todos os campos fornecidos serão dados de maneira regular, não havendo a necessidade de qualquer pré-processamento da entrada. A função deve falhar caso haja a tentativa de inserir um corredor com um `id_corredor` já cadastrado, ou seja, um CPF que já esteja no sistema. 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`.

Lembre-se de atualizar todos os índices necessários durante a inserção.

3.2 Remoção de corredores

```
DELETE FROM corredores WHERE id_corredor = '<id_corredor>';
```

O usuário deverá ser capaz de remover uma conta dado um CPF de um corredor. 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 saldo na conta

```
UPDATE corredor SET saldo = saldo + '<valor>' WHERE id_corredor = '<id_corredor>';
```

O usuário deverá ser capaz de adicionar valor na conta de um corredor dado seu CPF e o valor desejado. Caso o corredor não esteja cadastrado no sistema, o programa deve 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, o saldo deverá ser atualizado, seguido da impressão da mensagem padrão de `SUCESSO`.

3.4 Comprar e atribuir veículo a um corredor

```
UPDATE corredor SET veiculos = array_append(veiculos, '<veiculos>') WHERE
id_corredor = '<id_corredor>';
```

O usuário poderá adicionar um veículo a um corredor dado seu CPF e o modelo de veículo desejado (caso possua saldo para comprá-lo). Neste caso, o saldo será descontado da conta do jogador baseado no valor do veículo. Caso o corredor não possua saldo, a mensagem padrão `ERRO_SALDO_NAO_SUFICIENTE` deve ser impressa. Caso o corredor não exista, o programa deve imprimir `ERRO_REGISTRO_NAO_ENCONTRADO` e, caso o veículo novo já esteja presente nos veículos do corredor, seu programa deve imprimir `ERRO_VEICULO_REPETIDO`. Existe um máximo de três modelos de veículos

por corredor e, garantidamente, não haverá nenhuma tentativa de inserir mais de três por corredor. Caso não haja nenhum erro, o programa deve atribuir o veículo ao corredor, atualizando todos os índices e arquivos necessários e, então, imprimir a mensagem padrão `SUCESSO`.

3.5 Cadastro de veículos

```
INSERT INTO veiculos VALUES ('<marca>', '<modelo>', '<poder>', '<velocidade>',
'<aceleracao>', '<peso>', '<preco>');
```

Para um veículo ser adicionado no banco de dados, seu programa deverá ler os campos que contém a marca, modelo, poder, velocidade, aceleração, peso e preço. O campo 'id_veiculo 'segue a ordem de cadastro dos veículos. A função deve falhar caso haja a tentativa de inserir um veículo cujo ID já esteja presente no banco de dados, e deverá ser apresentada a mensagem de erro padrão `ERRO_PK_REPETIDA`. Caso a operação se concretize, exiba a mensagem padrão `SUCESSO`.

3.6 Cadastrar pista

```
INSERT INTO pistas VALUES ('<nome>', '<dificuldade>', '<distancia>', '<recorde>');
```

Para criar uma nova pista, seu programa deve ler os campos `nome`, `dificuldade`, `recorde` e `distancia`. O campo `id_pista` deverá ser preenchido de acordo com a quantidade de cursos cadastrados no sistema, ou seja, é um valor incremental. O único campo que é opcional quanto ao fornecimento é a `dificuldade`, que, caso não informado, receberá o valor padrão 1. Quando a operação se concretizar com sucesso, exibir a mensagem padrão `SUCESSO`.

3.7 Executar corrida

```
INSERT INTO corridas VALUES ('<id_pista>', '<ocorrencia>', '<id_corredores>',
'<id_veiculos>');
```

Para executar uma nova corrida, seu programa deve ler os campos `id_pista`, `ocorrencia`, `id_corredores` e `id_veiculos`. Todos os campos serão fornecidos de maneira regular, não havendo a necessidade de qualquer pré-processamento da entrada. Os identificadores de corredores no campo `id_corredores` estarão ordenados pela sua colocação, e os três primeiros corredores receberão uma premiação monetária. O prêmio `P` é calculado pela seguinte expressão: `P = 6 * (T * D)`, sendo `T` uma taxa fixa, `D` a dificuldade da pista em que a corrida foi executada e `(T * D)` a taxa de inscrição dessa pista. Nesse sentido, o primeiro colocado receberá 40% de `P`, o segundo colocado receberá 30% de `P` e o terceiro receberá 20% de `P`. Os 10% restantes serão reservados como taxa de manutenção do evento. Quando a operação se concretizar com sucesso, exibir a mensagem padrão `SUCESSO`.

3.8 Busca

As seguintes operações de busca por corredores e pistas 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: 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., 5a ou 6a 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 Corredores

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

(a) ID do corredor:

```
SELECT * FROM corredores WHERE id_corredor = '<id_corredor>';
```

Solicitar ao usuário o ID de cadastro do corredor. 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 Pistas

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

(a) ID da pista:

```
SELECT * FROM pistas WHERE id_pista = '<id_pista>';
```

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

(b) Pelo nome da pista:

```
SELECT * FROM pistas WHERE nome = '<nome da pista>';
```

Caso a pista não seja encontrada, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso contrário, todos os dados deverão ser impressos na tela de forma formatada. Observe que, neste caso, como a busca será realizada em dois índices, será necessário exibir o caminho da busca binária para ambos: primeiro para o índice secundário e depois para o índice primário.

3.9 Listagem

As seguintes operações de listagem deverão ser implementadas.

3.9.1 Corredores

(a) Pelos IDs dos corredores:

```
SELECT * FROM corredores ORDER BY id_corredor ASC;
```

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

(b) Por modelo de veículo:

```
SELECT * FROM veiculos WHERE '<veiculo>'= ANY (veiculos) ORDER BY id_veiculo ASC
```

Exibe todos os corredores que possuem determinado modelo de veículo, 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 corredores, o seu programa deverá imprimir os registros do índice da lista invertida (i.e., `corredor_veiculos_primario_idx`, ver detalhes na Seção 4.2).

3.9.2 Veículos para comprar

(a) De acordo com o saldo do corredor:

```
SELECT * FROM veiculos WHERE preco <= ('SELECT saldo FROM corredores WHERE
id_corredor = <id_corredor> ');
```

Seu programa deve ler o ID de um corredor e, em seguida, exibir todos os veículos que o corredor pode comprar, de acordo com o seu saldo. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

3.9.3 Corrida

(a) Por data de corrida:

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

Exibe todas as corridas realizadas em um determinado período de tempo (`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 corridas realizadas no período, primeiro é necessário imprimir o caminho percorrido durante a busca binária para encontrar o registro cuja `<ocorrencia>` seja igual à `<data_inicio>` informada pelo usuário ou data posterior mais próxima.

3.10 Liberar espaço

```
VACUUM corredores;
```

O arquivo de dados `ARQUIVO_CORREDORES` 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 corredores:

```
\echo file ARQUIVO_CORREDORES
```

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

(b) Dados dos cursos:

```
\echo file ARQUIVO_VEICULOS
```

Imprime o arquivo de dados de veículos. Caso esteja vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

(c) Dados das pistas:

```
\echo file ARQUIVO_PISTAS
```

Imprime o arquivo de dados de pistas. Caso o arquivo esteja vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

(d) Dados das corridas:

```
\echo file ARQUIVO_CORRIDAS
```

Imprime o arquivo de corridas realizadas. Caso o arquivo esteja 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 corredores com `id_corredor` e `rrn`:

```
\echo index corredores_idx
```

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

(b) Índice de veículos com `id_veiculo` e `rrn`:

`\echo index veiculos_idx`

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

(c) Índice de pistas com `id_pista` e `rrn`:

`\echo index pistas_idx`

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

(d) Índice de corridas com `ocorrencia`, `id_pista` e `rrn`:

`\echo index corridas_idx`

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

3.13 Imprimir índices secundários

O sistema deverá imprimir os índices secundários da seguinte maneira:

(a) Índice de nome pista com `nome` e `id_pista`:

`\echo index nome_pista_idx`

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

(b) Índice de preço com `preco` e `id_veiculo`:

`\echo index preco_veiculo_idx`

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

(c) Índice de corredor veículos secundário:
`\echo index corredor_veiculos_secundario_idx`

Imprime as *structs* de índice secundário com os modelos de veículos (`corredor_veiculos_secundario_idx`) e o número de índice para o primeiro corredor proprietário desse modelo. Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(d) Índice de corredor veículos primário (obs: no escopo deste trabalho, este índice também é secundário):
`\echo index corredor_veiculos_primario_idx`

Imprime as *structs* de índice secundário com o ID de um corredor proprietário de um modelo de veículo (`corredor_veiculos_primario_idx`) e o número de índice para o próximo corredor proprietário. Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

3.14 Finalizar

`\q`
Libera a memória e encerra a execução do programa.

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

- `corredores_idx`: índice primário que contém o ID do corredor (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pelo ID do corredor (`id_corredor`);
- `veiculos_idx`: índice primário que contém o ID do veículo (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pelo ID do veículo (`id_veiculo`);
- `pistas_idx`: índice primário que contém o ID da pista (chave primária) e o RRN respectivo do registro no arquivo de pistas, ordenado pelo ID da pista (`id_pista`);
- `corridas_idx`: índice primário que consiste na ocorrência (data e horário) da corrida, o ID da pista em que os corredores se inscreveram e o RRN relativo ao registro no arquivo de corridas, ordenado pela ocorrência (`ocorrencia`) e o ID da pista (`id_pista`).

4.2 Índices secundários

- `nome_pista_idx`: índice secundário que contém as pistas ordenadas por nome e a chave primária (`id_pista`) da pista específica.
- `preco_veiculo_idx`: índice secundário que contém os veículos ordenados por preço e a chave primária (`id_veiculo`) do veículo específico.
- `corredor_veiculos_idx`: índice secundário do tipo *lista invertida*. Será necessário manter dois índices (`corredor_veiculos_primario_idx` e `corredor_veiculos_secundario_idx`), sendo que o primário possui os IDs de corredores (`id_corredor`) que possuem certo modelo de veículo

e o apontador para o próximo corredor proprietário do modelo nesse mesmo índice primário. Se não houver um próximo corredor, esse apontador deve possuir o valor -1. No índice secundário estão os modelos, assim como a referência do primeiro corredor proprietário daquele modelo no índice primário.

Para simplificar o entendimento, considere o seguinte exemplo:

Corredor veículos primário	
ID corredor	próx. registro
0	4
4	-1
4	7
4	5
3	-1
7	6
5	-1
5	-1
...	...

Corredor veículos secundário	
Modelo	registro
RORIMAN KATO	0
STANDARD UFSKART	1
TURING FUMACA	3
BUGGY WUGGY	2
...	...

No exemplo acima, a tabela de corredor veículos secundário possui o modelo na primeira coluna, assim como o RRN do primeiro corredor proprietário daquele modelo que foi inserido na tabela de corredor veículos primário. Na tabela primária, tem-se na primeira coluna o ID dos corredores proprietários de cada modelo. Note que o corredor com ID = 4 aparece três vezes no exemplo, o que significa que ele possui três modelos de veículos diferentes. Na segunda coluna da tabela primária, temos o RRN para o próximo corredor proprietário de um mesmo modelo na própria tabela de corredor veículos primária, sendo que, RNN = -1, significa que aquele corredor é o último que possui o mesmo modelo. 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 impresso 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_CORREDORES`, `ARQUIVO_VEICULOS`, `ARQUIVO_PISTAS` e `ARQUIVO_CORRIDAS`, 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).

`ARQUIVO_CORREDORES`: deverá ser organizado em registro de tamanho fixo de 160 *bytes* (160 caracteres). Os campos `nome` (tamanho máximo de 44 *bytes*), `apelido` (tamanho máximo de 40 *bytes*) e `veiculo` (máximo de 45 *bytes*, com no máximo 3 valores, onde cada modelo pode ter no máximo 14 *bytes*) devem ser de tamanho variável. O campo multi-valorado `veiculo` deve ter os seus valores separados por '`|`'. Os demais campos devem ser de tamanho fixo, sendo eles `id_corredor` (11 *bytes*), `cadastro` (12 *bytes*) e `saldo` (13 *bytes*). Portanto, os campos de tamanho fixo de um registro ocuparão 36 *bytes*. Os campos devem ser separados pelo caractere delimitador '`;`' (ponto e vírgula), e cada registro terá 6 delimitadores (um para cada campo). Caso o registro tenha menos de 149 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 36 *bytes* fixos + 6 *bytes* de delimitadores, então os campos variáveis devem ocupar no máximo 118 *bytes*, para que o registro não exceda os 149 *bytes*.

Exemplo de arquivo de dados de corredores:

```
12345678910;Arnaldo Turbilhao;Turbinaldo;202309251000;0000003000.00;
hot-dog|marinho|;####################################################
########################92345678915;Flavio dos Santos;Flavinho do Pn
eu;202305121552;0000005200.00;borracha|;############################
####################################################09898989999;Catia So
uza;Cata-vento;202307022100;0000075000.00;aeroplano|teco-teco|;#####
#####################################################################
####10111213141;Mefistofelio Credolfo de Assis;Mefisto running;20190
3201400;000000987.65;adornado|simplorio|pancadao|;##################
############################
```

`ARQUIVO_VEICULOS`: o arquivo de veículos deverá ser organizado em registros de tamanho fixo de 128 *bytes* (128 caracteres). Os campos `marca` (máximo de 23 *bytes*), `modelo` (máximo de 14 *bytes*) e `poder` (máximo de 51 *bytes*) devem ser de tamanhos variáveis. Os demais campos são de tamanho fixo e possuem as seguintes especificações: `id_veiculo` (7 *bytes*), `velocidade` (4 *bytes*), `aceleracao` (4 *bytes*), `peso` (4 *bytes*) e `valor` (13 *bytes*), totalizando 32 *bytes* de tamanho fixo. Assim como no registro de corredores, os campos devem ser separados pelo delimitador '`;`', cada registro terá 8 delimitadores para os campos e, caso o registro tenha menos que 124 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 32 *bytes* de tamanho fixo + 8 *bytes* para os delimitadores, os campos variáveis devem ocupar no máximo 88 *bytes* para que não se exceda o tamanho do registro.

Exemplo de arquivo de dados de veículos:

```
1230400;Pevrolet;Fliperama Furioso;Armas retro estilo 8 bit;0060;0010;
0200;0000000199.99;########################################4690901;Tiaf
;Turing Fumaca;Cortina de fumaca binaria;9999;001;0753;0000022199.87;#
##############################################5474102;Lesta;Model ML;P
iloto automatico;0070;0015;0353;0000122199.87#######################
#################################
```

`ARQUIVO_PISTAS`: o arquivo de pistas deverá ser organizado em registros de tamanho fixo de 56 *bytes* (56 caracteres). Apenas o campo `nome` (máximo de 31 *bytes*) deve ser de tamanho variável. Os demais campos são de tamanho fixo e possuem as seguintes especificações: `id_pista` (8 *bytes*), `dificuldade` (4 *bytes*), `distancia` (4 *bytes*) e `recorde` (4 *bytes*), totalizando 20 *bytes* de tamanho fixo. Assim como nos outros registros citados, os campos devem ser separados pelo delimitador '`;`', cada registro terá 5 delimitadores para os campos e, caso o registro tenha menos que 56 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 20 *bytes* de tamanho fixo + 5 *bytes* para os delimitadores, os campos variáveis devem ocupar no máximo 31 *bytes* para que não se exceda o tamanho do registro.

Exemplo de arquivo de dados de pistas:

```
00000000;Star Road;0002;2500;1100;######################
00000001;SoroCaos;0003;3000;6310;######################
00000002;Circuito de apesentacao: Tour;0001;0500;0100;##
00000003;Corrida pos-aula: onibus;0003;0500;0070;#######
00000004;Maratona pre-prova;0003;9999;0000;#############
00000005;Yoshi's Valley;0002;0750;2175;#################
```

`ARQUIVO_CORRIDAS`: o arquivo de corridas deverá ser organizado em registros de tamanho fixo de 128 *bytes*. Todos os campos possuem um tamanho fixo e, portanto, não é necessário a presença de delimitadores: `id_pista` (8 *bytes*), `ocorrencia` (12 *bytes*), `id_corredores` (66 *bytes*), `id_veiculos` (42 *bytes*)). No exemplo abaixo, os campos estão ordenados na mesma sequência apresentada acima.

Exemplo de arquivo de dados de corridas:

```
00000000 202303111717 37567876542 58567446542
46567876542 00567876541 34567810567 12567876542
0000000 0000001 0000002 0000003 00000004 0000005

00000000 202312312359 31387876542 78767446500
76567811549 00567876541 34567810563 12567876540
0000000 0000006 0000007 0000008 00000009 0000001

00000004 202310051617 11387876542 58567446542
46567876542 00567876541 34005678670 12567876542
0000100 0000016 0000004 0000018 00000049 0000009
```

No exemplo acima, foram inseridos espaços em branco entre os campos apenas para maior clareza do exemplo. Note também, que não há quebras de linhas nos arquivos (elas foram inseridas apenas para facilitar a visualização da sequência de registros).

6 Instruções para as operações com os registros

- **Inserção**: cada corredor, veículo, pista e corrida devem ser inseridos no final de seus respectivos arquivos de dados, e atualizados os índices.
- **Remoção**: o registro de um dado corredor deverá ser localizado acessando o índice primário (`id_corredor`). 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 160 *bytes*. Além disso, no índice primário, o RRN correspondente ao registro removido deverá ser substituído por -1.
- **Atualização**: existe um campo alterável: (i) o saldo do corredor. Eles possui 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_CORREDORES TO '<DADOS DE CORREDORES>';` caso queira inicializar o programa com um arquivo de corredores já preenchido;
2. Inserir o comando `SET ARQUIVO_VEICULOS TO '<DADOS DE VEICULO>';` caso queira inicializar o programa com um arquivo de veículos já preenchido;
3. Inserir o comando `SET ARQUIVO_PISTAS TO '<DADOS DE PISTA>';` caso queira inicializar o programa com um arquivo de pistas já preenchido;
4. Inserir o comando `SET ARQUIVO_CORRIDAS TO '<DADOS DE CORRIDA>';` caso queira inicializar o programa com um arquivo de corrida 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_corredor(int rrn)`, `exibir_veiculo(int rrn)`, `exibir_pista(int rrn)` ou `exibir_corrida(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

- 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.

"A imaginação é mais importante que o conhecimento"
— Albert Einstein