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: 10/03/2022 – 08:00

Atenção

- **Sistema de submissão:** http://judge.sor.ufscar.br/ori/
- **Arquivo:** deverá ser submetido um único código-fonte seguindo o padrão de nomenclatura `<RA>_ORI_T01.c`, ex: `123456_ORI_T01.c`;
- **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

A indústria de jogos vem crescendo exponencialmente nos últimos anos, partindo de mercados focais até atingir o seu imenso tamanho que pode ser observado hoje em dia. Só em 2021, a área movimentou mais de 93,2 bilhões de dólares e contabilizou cerca de 3 bilhões de jogadores pelo mundo. Diversas gerações de consoles foram comercializados desde a década de 70 e, embora seja considerada uma indústria relativamente nova, diversas inovações já foram apresentadas com o passar dos anos.

Hoje em dia, existe uma grande facilidade no acesso aos jogos, desde aqueles produzidos por grandes empresas até aos de criadores independentes. Tamanha facilidade deve-se às plataformas virtuais de distribuição, como por exemplo a Steam, a Epic games Store, o itch.io, entre outras. Todas essas plataformas precisam agregar dados de seus usuários e dos desenvolvedores de jogos, para que as compras possam ser efetivadas e as pessoas consigam seus tão sonhados jogos.

Observando de perto o grande sucesso que essas plataformas apresentam e o quão promissor esse mercado parece ser, você teve a brilhante ideia de desenvolver a sua própria plataforma de distribuição de jogos: a **UFSPlay**, com foco em distribuição de jogos nacionais e internacionais. Contudo, antes de botar esse grande projeto em prática, será necessário criar um MVP, produto mínimo viável dessa ideia. Para isso, você precisará empregar suas excelentes habilidades na linguagem C para criar e manipular de forma eficiente as bases de dados dos jogos e dos usuários dessa nova plataforma.

2 Base de dados da aplicação

O sistema será composto por dados dos usuários, dos jogos e das compras, conforme descrito a seguir.

2.1 Dados do usuário

- **ID_User**: identificador único de um usuário (chave primária) que contém os 11 números do ID. Não poderá existir outro valor idêntico na base de dados. Ex: `57956238064`;
- **Username**: nome escolhido pelo usuário. Ex: `xXxSN1P3RxXx`;
- **Email**: e-mail do usuário. Ex: `tomas.aquino@gmail.com`;
- **Celular**: número de contato no formato `<99><999999999>`. Ex: `15924835754`;
- **Saldo**: saldo do usuário no formato `<9999999999>.<99>`. Ex: `0000004605.10`;

2.2 Dados do jogo

- **ID_Game**: Um identificador numérico único para o jogo que pode conter até 8 dígitos. Ex: `01234567`;
- **Título**: título do jogo, que deve ser único;
- **Desenvolvedor**: nome do desenvolvedor do jogo;
- **Editora**: nome da editora do jogo;

- **Lançamento**: Data em que o jogo foi lançado no formato `<AAAA><MM><DD>`. Ex: `20170224`;
- **Preço**: Preço do jogo no formato `<99999999999>.<99>`. Ex: `00000000027.99`
- **Categoria**: Até três categorias que o jogo pode pertencer, separadas por um '|'. Ex: `Ação|Aventura|Plataforma`

2.3 Dados da compra

- **ID_User**: ID do usuário que está realizando a compra do jogo;
- **ID_Game**: ID do jogo comprado;
- **Data**: Data em que a compra do jogo foi realizada no formato `<AAAA><MM><DD>`.
  Ex: `20220102`;

Garantidamente, nenhum campo de texto receberá caracteres acentuados.

2.4 Modelo relacional e DDL

A base de dados será mantida em forma de arquivos, porém se fosse visualizada em um modelo relacional, seria equivalente ao observado na Figura 1.

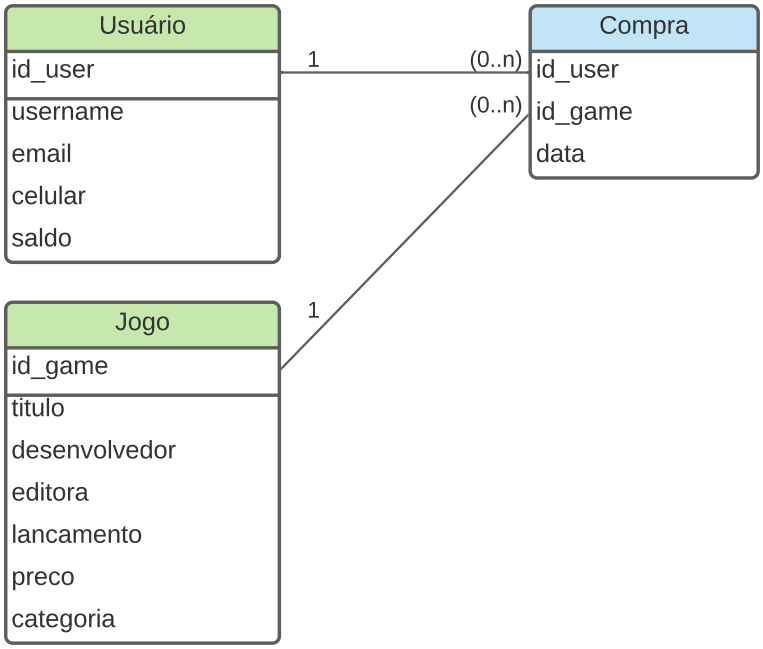

Figura 1: Modelo relacional das tabelas de usuários, jogos e compras da UFSPlay

```
CREATE TABLE usuarios (
  id_user   varchar(11) NOT NULL PRIMARY KEY,
  username  text NOT NULL,
  email     text NOT NULL,
  celular   varchar(11),
  saldo     numeric(12, 2) DEFAULT 0,
);
```

```
CREATE TABLE jogos (
  id_game        varchar(8) NOT NULL PRIMARY KEY,
  titulo         text NOT NULL UNIQUE,
  desenvolvedor  text NOT NULL,
  editora        text NOT NULL,
  lancamento     varchar(8) NOT NULL,
  preco          numeric(12, 2) NOT NULL,
  categoria      text[3] DEFAULT '{}'
);
CREATE TABLE compras (
  id_user  varchar(11) NOT NULL,
  id_game  varchar(8) NOT NULL,
  data     varchar(8) NOT NULL,
);
```

3 Operações suportadas pelo programa

Deve ser possível interagir com o programa 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 usuário

```
INSERT INTO usuarios VALUES ('<id_user>', '<username>', '<email>');
```

Para criar uma nova conta de usuário, seu programa deve ler os campos `id_user`, `username` e `email`. Inicialmente, a conta será criada sem saldo (R$0,00) e sem um número de celular, sendo que este deverá ser preenchido com '*' (asterisco) para todos os dígitos do número. 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 usuário com ID_User já cadastrado, ou seja, com ID_User 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 usuário

```
DELETE FROM usuarios WHERE id_user = '<id_user>';
```

O usuário deverá ser capaz de remover uma conta dado o ID_User da mesma. 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 se concretizar, exiba a mensagem padrão `SUCESSO`.

3.3 Cadastro de jogo

```
INSERT INTO jogos VALUES ('<titulo>', '<desenvolvedor>', '<editora>', '<lanca-
mento>', '<preco>');
```

O jogo é adicionado no banco de dados. Seu programa deve ler os campos que contém o título, desenvolvedor, editora, lançamento e preço. Assim como no cadastro de usuário, todos os campos serão fornecidos de maneira regular, não havendo a necessidade de pré-processamento. O ID_Game do jogo deverá ser preenchido de acordo com a quantidade de jogos cadastrados no sistema, ou seja, é um valor incremental. A função deve falhar caso haja a tentativa de inserir um jogo cujo título já esta 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`.

3.4 Compra de jogo

```
INSERT INTO compras VALUES ('<id_user>', '<titulo>');
```

O usuário poderá comprar um jogo dado o ID_User do usuário e o título do jogo. Caso o jogo ou o usuário não existam, o programa deve imprimir `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso a compra já tenha sido realizada, o seu programa deve imprimir `ERRO_PK_REPETIDA`. Primeiro, é preciso checar se o jogo e o usuário existem. Caso o saldo presente na conta do usuário seja insuficiente para comprar o jogo, ela não deve ser efetuada e a mensagem padrão `ERRO_SALDO_NAO_SUFICIENTE` deve ser imprimida. Caso não haja nenhum desses problemas, a compra deve ser realizada, gravando a data em que a foi feita, atualizando os índices necessários e imprimindo a mensagem padrão `SUCESSO`.

3.5 Cadastro de celular

```
UPDATE usuarios SET celular = '<celular>' WHERE id_user = '<id_user>';
```

O usuário deverá poder atualizar seu número de celular dado o ID_User da conta e o número novo. Caso o usuário não esteja cadastrado no sistema, o programa deve imprimir a mensagem `ERRO_REGISTRO_NAO_ENCONTRADO`. Senão, o programa deve atualizar o telefone do usuário e imprimir a mensagem padrão `SUCESSO`.

3.6 Adicionar saldo na conta

```
UPDATE usuarios SET saldo = saldo + '<valor>' WHERE id_user = '<id_user>';
```

O usuário deverá ser capaz de adicionar dinheiro na sua conta dado seu ID_User e o valor desejado. Caso o usuário 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`.

3.7 Atribuir categoria a um jogo

`UPDATE jogos SET categorias = array_append(categorias, '<categorias>') WHERE titulo = '<titulo do jogo>';`

O usuário deve poder adicionar uma categoria a um jogo dado seu título e a categoria escolhida. Caso o jogo não exista, o programa deve imprimir `ERRO_REGISTRO_NAO_ENCONTRADO` e caso a categoria nova já esteja presente nas categorias do jogo, seu programa deve imprimir `ERRO_CATEGORIA_REPETIDA`. Existe um máximo de três categorias por jogo e, garantidamente, não haverá nenhuma tentativa de inserir mais de três por jogo. Caso não haja nenhum erro, o programa deve atribuir a categoria ao jogo, atualizando todos os índices e arquivos necessários e então imprimir a mensagem padrão `SUCESSO`.

3.8 Busca

As seguintes operações de busca por usuários e jogos deverão ser implementadas. *Em todas elas, será necessário utilizar a busca binária e mostrar o caminho percorrido 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.

3.8.1 Usuários

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

(a) Por ID_User de usuário:

`SELECT * FROM usuarios WHERE id_user = '<id_user>';`

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

3.8.2 Jogos

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

(a) Pelo ID_Game do jogo:

`SELECT * FROM jogos WHERE id_game = '<id_game>';`

Solicitar ao usuário o ID_Game do jogo no formato correto de 8 dígitos. Caso o jogo não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o jogo exista, todos os dados do jogo deverão ser impressos na tela de forma formatada.

(b) Pelo título do jogo:

```
SELECT * FROM jogos WHERE titulo = '<titulo do jogo>';
```

Caso o título do jogo não seja encontrado, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o jogo exista, todos os dados deverão ser impressos na tela de forma formatada.

3.9 Listagem

As seguintes operações de listagem de usuários e compras deverão ser implementadas.

3.9.1 Usuários

(a) Por ID_User dos usuários:

```
SELECT * FROM usuarios ORDER BY id_user ASC;
```

Exibe todos os usuários ordenados de forma crescente pelo ID_User. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

3.9.2 Jogos

(a) Por categoria:

```
SELECT * FROM jogos WHERE '<categoria>'= ANY (categorias) ORDER BY id_game ASC
```

Exibe todos os jogos de uma determinada categoria, em ordem crescente de ID_Game. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

Antes da listagem dos jogos, o seu programa deve imprimir os registros do índice secundário que foram percorridos na listagem (lembrando que a busca no índice secundário que contém as categorias, deverá ser feito por uma busca binária!). Exemplo:

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

No exemplo acima, os números representam o numero do índice dos registros que foram percorridos durante a busca até encontrar o registro de interesse ou esgotar as possibilidades no índice secundário como foi previamente dito.

3.9.3 Compras

(a) Por data de aquisição do jogo:

```
SELECT * FROM compras WHERE data_compra BETWEEN '<data_inico>' AND '<data_fim>'
ORDER BY data_compra ASC;
```

Exibe todas as compras realizada em um determinado período de tempo (`data` entre `<data início>` e `<data fim>`), em ordem cronológica. Cada registro encontrado na listagem deve imprimir seu caminho percorrido. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

3.10 Liberar espaço

```
VACUUM usuarios;
```

O arquivo de dados `ARQUIVO_USUARIOS` 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 usuários:

```
\echo file ARQUIVO_USUARIOS
```

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

(b) Dados dos jogos:

```
\echo file ARQUIVO_JOGOS
```

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

(c) Dados das compras:

```
\echo file ARQUIVO_COMPRAS
```

Imprime o arquivo de compras feitas. 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 usuários com `id_user` e `rrn`:

```
\echo index usuarios_idx
```

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

(b) Índice de jogos com `id_game` e `rrn`:

`\echo index jogos_idx`

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

(c) Índice de compras com `id_user`, `id_game` e `rrn`:

`\echo index compras_idx`

Imprime as *structs* de índice primário de compras. 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 títulos com `titulo` e `id_game`:

`\echo index titulo_idx`

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

(b) Índice de datas de compras com `data`, `id_user` e `id_game`:

`\echo index data_user_game_idx`

Imprime as *structs* de índice secundário de datas das compras. Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(c) Índice de categorias secundário:
`\echo index categorias_secundario_idx`

Imprime as *structs* de índice secundário com as categorias dos jogos (`categorias_secundario_idx`) e o número de índice para o primeiro jogo da categoria. Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

(d) Índice de categorias primário (no escopo do trabalho em geral, este índice é secundário):
`\echo index categorias_primario_idx`

Imprime as *structs* de índice secundário com o ID_Game do jogo em uma categoria (`categorias_primario_idx`) e o número de índice para o próximo jogo desta categoria. 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).

Pelo menos os seguintes índices deverão ser criados:

4.1 Índices primários

- `usuarios_idx`: índice primário que contém o ID_User do usuário (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pelo ID_User de forma crescente;
- `jogos_idx`: índice primário que contém o ID_Game do jogo (chave primária) e o RRN respectivo do registro no arquivo de jogos, ordenado pelo ID_Game de forma crescente;
- `compras_idx`: índice primário que consiste no ID_User do usuário que realizou a compra, o ID_Game do jogo comprado e o RRN relativo ao registro no arquivo de compras, ordenado pelo ID_User e o ID_Game de forma crescente.

4.2 Índices secundários

- `titulo_idx`: índice secundário que contém os jogos ordenados por títulos (em ordem lexicográfica) e a chave primária (ID_Game) do jogo específico.
- `data_user_game_idx`: índice secundário que contém as datas das compras (em ordem cronológica), o ID_User do usuário (em ordem lexicográfica) que realizou aquela compra e o ID_Game do jogo comprado.
- `categorias_idx`: índice secundário do tipo *lista invertida*. Será necessário manter dois índices (`categorias_primario_idx` e `categorias_secundario_idx`), sendo que o primário possui os ID_Game de um jogo da categoria e o apontador para o próximo jogo da mesma categoria nesse mesmo índice primário. Se não houver um próximo jogo, esse apontador deve possuir o valor -1. No índice secundário estão as categorias, assim como a referência do primeiro jogo daquela categoria no índice primário.

  Para simplificar o entendimento, considere o seguinte exemplo:

Categorias primário	
ID_Game do jogo	próx. registro
0	4
4	-1
4	7
4	5
3	-1
7	6
5	-1
5	-1

Categorias secundário	
Categoria	registro
Acao	3
FPS	0
Metroidvania	1
Plataforma	2

No exemplo acima, podemos observar que a tabela de categorias secundário possui a categoria na primeira coluna, assim como o RRN do primeiro jogo daquela categoria que foi inserido na tabela de categorias primário. Na tabela primária, temos na primeira coluna o ID_Game dos jogos em cada categoria. Note que o jogo com ID = 4 aparece três vezes no exemplo, o que significa que ele pertence a três categorias diferentes. Na segunda coluna da tabela primária, temos o RRN para o próximo jogo de mesma categoria na própria tabela de categorias primária, sendo que, RNN = -1, significa que aquele jogo é o último de sua categoria. 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. Os índices 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.

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_USUARIOS` , `ARQUIVO_JOGOS` e `ARQUIVO_COMPRAS`, 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_USUARIOS`: deverá ser organizado em registro de tamanho fixo de 128 *bytes* (128 caracteres). Os campos `username` (tamanho máximo de 47 *bytes*) e `email` (tamanho máximo de 41 *bytes*) devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo, sendo eles `id_user` (11 *bytes*), `celular` (11 *bytes*) e `saldo` (13 *bytes*). Portanto, os campos de tamanho fixo de um registro ocuparão 35 *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 128 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 35 *bytes* fixos + 5 *bytes* de delimitadores, então os campos variáveis devem ocupar no máximo 88 *bytes*, para que o registro não exceda os 128 *bytes*.

Exemplo de arquivo de dados de usuários:

```
66545678765;SnailThemi;mariaeugenia@gmail.com;**********;00000
00030.00;##################################################
##99876556782;ezPz#Fireclan;ga.augusto@gmail.com;**********;00
00000000.00;##################################################
####44565434213;forninhocaiu;ge.santana@gmail.com;**********;0
000000029.69;##################################################
######54654367865;batata_doce;galmeida@gmail.com;**********;00
00001250.80;##################################################
########37567876542;Erlkonigin;melissa@gmail.com;**********;10
00000440.92;##################################################
##########
```

`ARQUIVO_JOGOS`: o arquivo de jogos deverá ser organizado em registros de tamanho fixo de 256 *bytes* (256 caracteres). Os campos `titulo` (máximo de 43 *bytes*), `desenvolvedor` (máximo de 47 *bytes*), `editora` (máximo de 47 *bytes*) e `categoria` (máximo de 60 *bytes*, com no máximo 3 valores, onde cada categoria pode ter no máximo 20 *bytes*) devem ser de tamanhos variáveis. O campo multivalorado `categoria` deve ter os seus valores separados por '`|`'. Os demais campos são de tamanho fixo e possuem as seguintes especificações: `id_game` (8 *bytes*), `lancamento` (8 *bytes*) e `preco` (13 *bytes*), totalizando 29 *bytes* de tamanho fixo. Assim como no registro de usuários, os campos devem ser separados pelo delimitador '`;`', cada registro terá 7 delimitadores para os campos e mais 3 para o campo multivalorado e, caso o registro tenha menos que 256 *bytes*, o espaço restante deverá ser preenchido com o caractere '`#`'. Como são 29 *bytes* de tamanho fixo + 7 *bytes* para os delimitadores, os campos variáveis devem ocupar no máximo 220 *bytes* para que não se exceda o tamanho do registro.

Exemplo de arquivo de dados de jogos:

```
00000000;Meia-Vida;Valvula;Valvula;19981119;0000000029.99;FPS;###
############################################################
############################################################
############################################################0000
0001;Presa;Cabeca de Melao Studios;40K Martelos;20070711;00000000
44.29;;#####################################################
############################################################
############################################################00000002
;Quinze Minutos;Antonio Luis;Annapurr;20210819;0000000051.99;;###
############################################################
############################################################
############################################################00000003;Esq
uerda 4 Morto 1;Valvula;Valvula;20091117;0000000020.69;FPS;######
############################################################
############################################################
############################################################00000004;Cavalei
ro Vazio;Time Cereja;Time Cereja;20170224;0000000027.99;Metroidva
nia|Plataforma|Acao;########################################
############################################################
############################################
```

`ARQUIVO_COMPRAS`: o arquivo de compras deverá ser organizado em registros de tamanho fixo de 27 *bytes*. Todos os campos possuem um tamanho fixo e, portanto, não é necessário a presença de delimitadores: `id_user` (11 *bytes*), `id_game` (8 *bytes*) e `data` (8 *bytes*). No exemplo abaixo, os campos estão ordenados por `id_user`, `data` e `id_game`.

Exemplo de arquivo de dados de compras:

```
44565434213 20210920 00000000
37567876542 20210924 00000004
37567876542 20211003 00000008
66545678765 20211005 00000005
```

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 usuário, jogo e compra devem ser inseridos no final de seus respectivos arquivos de dados, e atualizados os índices.
- **Remoção**: o registro deverá ser localizado acessando o índice primário. A remoção deverá colocar o marcador `*|` nas 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 256 *bytes*. Além disso, no índice primário, o RRN correspondente ao registro removido deverá ser substituído por -1.

- **Atualização**: existem dois campos alteráveis, o saldo e o celular do usuário. Todos eles possuem seus tamanhos pré-determinados: 13 *bytes* para o saldo e 11 *bytes* para o celular. Esses dados devem ser atualizados 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_USUARIOS TO '<DADOS DE USUARIOS>';` caso queira inicializar o programa com um arquivo de usuários já preenchido;
2. Inserir o comando `SET ARQUIVO_JOGOS TO '<DADOS DE JOGOS>';` caso queira inicializar o programa com um arquivo de jogos já preenchido;
3. Inserir o comando `SET ARQUIVO_COMPRAS TO '<DADOS DE COMPRAS>';` caso queira inicializar o programa com um arquivo de compras já preenchido;
4. 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_usuario(int rrn)`, `exibir_jogo(int rrn)` ou `exibir_compra(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.

"Respire. Você consegue."
— Madeline, Celeste