

**Universidade de São Paulo  
Instituto de Ciências Matemáticas e de Computação  
Departamento de Ciências de Computação  
Disciplina de Organização de Arquivos (SCC0215)**

Docente

Profa. Dra. Cristina Dutra de Aguiar Ciferri  
[cdac@icmc.usp.br](mailto:cdac@icmc.usp.br)

Colaborador e monitor (Turma 1 – segunda-feira)

João Paulo Clarindo  
[jpcsantos@usp.br](mailto:jpcsantos@usp.br)

Luis Eduardo Rozante de Freitas Pereira  
[luis.eduardo.rozante@usp.br](mailto:luis.eduardo.rozante@usp.br) ou telegram: @LuisEduardo\_Cabral

Monitores (Turma 2 – terça-feira)

Mateus Prado Santos  
[mateus.prado@usp.br](mailto:mateus.prado@usp.br)

Lucas de Medeiros Franca Romero  
[lucasromero@usp.br](mailto:lucasromero@usp.br) ou telegram: @lucasromero

**Segundo Trabalho Prático**

**Este trabalho tem como objetivo indexar arquivos de dados usando um índice árvore-B.**

*O trabalho deve ser feito por, no máximo, 2 alunos da mesma turma. Os alunos devem ser os mesmos alunos do primeiro trabalho prático. Caso haja mudanças, elas devem ser informadas para a docente e os monitores. A solução deve ser proposta exclusivamente pelo(s) aluno(s) com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.*

---

**Fundamentos da disciplina de Bases de Dados**

---

A disciplina de Organização de Arquivos é uma disciplina fundamental para a disciplina de Bases de Dados. A definição dos trabalhos práticos será feita considerando esse aspecto, ou seja, o projeto será especificado em termos de várias funcionalidades, e essas funcionalidades serão relacionadas com as funcionalidades da linguagem SQL (*Structured Query Language*), que é a linguagem utilizada por

sistemas gerenciadores de banco de dados (SGBDs) relacionais. As características e o detalhamento de SQL serão aprendidos na disciplina de Bases de Dados. Porém, por meio do desenvolvimento dos trabalhos práticos, os alunos poderão entrar em contato desde a disciplina de Organização de Arquivos com alguns comandos da linguagem SQL e verificar como eles são implementados.

Este trabalho prático tem como objetivo relacionar o conceito de índice.

---

### Descrição de páginas de disco

---

No trabalho será usado o conceito de páginas de disco. Cada página de disco tem o tamanho fixo de 77 bytes. O conceito de página de disco é um conceito lógico, ou seja, deve ser garantido via programação, de forma que cada página de disco contenha, no máximo, o tamanho fixo especificado.

**Importante.** Na prática, o tamanho de página de disco do arquivo de dados e do arquivo de índice árvore-B é o mesmo. Entretanto, não foi definido um tamanho de página de disco para os arquivos de dados no primeiro trabalho prático. Dessa forma, os arquivos já implementados até então podem ser utilizados sem necessidade de alteração.

---

### Descrição do arquivo de índice árvore-B

---

O índice árvore-B com ordem  $m$  é definido formalmente como descrito a seguir.

1. Cada página (ou nó) do índice árvore-B deve ser, pelo menos, da seguinte forma:

$\langle P_1, \langle C_1, P_{R1} \rangle, P_2, \langle C_2, P_{R2} \rangle, \dots, P_{q-1}, \langle C_{q-1}, P_{Rq-1} \rangle, P_q \rangle$ , onde  $(q \leq m)$  e

- Cada  $P_j$  ( $1 \leq j \leq q$ ) é um ponteiro para uma subárvore ou assume o valor -1 caso não exista subárvore.
- Cada  $C_i$  ( $1 \leq i \leq q - 1$ ) é uma chave de busca.
- Cada  $P_{Ri}$  ( $1 \leq i \leq q - 1$ ) é um campo de referência para o registro no arquivo de dados que contém o registro de dados correspondente a  $C_i$ .

2. Dentro de cada página (ou seja, as chaves de busca são ordenadas)
  - $C_1 < C_2 < \dots < C_{q-1}$ .
3. Para todos os valores  $X$  da chave na subárvore apontada por  $P_i$ :
  - $C_{i-1} < X < C_i$  para  $1 < i < q$
  - $X < K_i$  para  $i = 1$
  - $K_{i-1} < X$  para  $i = q$ .
4. Cada página possui um máximo de  $m$  descendentes.
5. Cada página, exceto a raiz e as folhas, possui no mínimo  $\lceil m/2 \rceil$  descendentes (*taxa de ocupação*).
6. A raiz possui pelo menos 2 descendentes, a menos que seja um nó folha.
7. Todas as folhas aparecem no mesmo nível.
8. Uma página não folha com  $k$  descendentes possui  $k-1$  chaves.
9. Uma página folha possui no mínimo  $\lceil m/2 \rceil - 1$  chaves e no máximo  $m - 1$  chaves (*taxa de ocupação*).

**Descrição do Registro de Cabeçalho.** O registro de cabeçalho deve conter os seguintes campos:

- *status*: indica a consistência do arquivo de índice, devido à queda de energia, travamento do programa, etc. Pode assumir os valores 0, para indicar que o arquivo de dados está inconsistente, ou 1, para indicar que o arquivo de dados está consistente. Ao se abrir um arquivo para escrita, seu status deve ser 0 e, ao finalizar o uso desse arquivo, seu status deve ser 1 – tamanho: *string* de 1 byte.
- *noRaiz*: armazena o RRN do nó (página) raiz do índice árvore-B. Quando a árvore-B está vazia, *noRaiz* = -1 – tamanho: inteiro de 4 bytes.
- *RRNproxNo*: armazena o RRN do próximo nó (página) do índice árvore-B a ser inserido. Assume inicialmente o valor 0, sendo esse valor incrementado a cada criação de um novo nó – tamanho: inteiro de 4 bytes.

**Representação Gráfica do Registro de Cabeçalho.** O tamanho do registro de cabeçalho deve ser de 77 bytes, representado da seguinte forma:

1 byte	4 bytes	4 bytes	68 bytes
<i>status</i>	<i>noRaiz</i>	<i>RRNproxNo</i>	<i>lixo identificado pelo caractere '@'</i>
0	1...4	5...8	9...76

### Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Os nomes dos atributos também devem seguir estritamente os nomes definidos na especificação dos mesmos.
- Para seguir a especificação do conceito de árvore-B, o nó da árvore-B deve obrigatoriamente ser do tamanho de uma página de disco. Entretanto, isso não será seguido neste trabalho para simplificar a quantidade de chaves de busca que são armazenadas no nó. Lembrando também que as páginas de disco têm potência de 2, o que também não será seguido neste trabalho por simplificação.
- Os 68 bytes restantes devem ser preenchidos com lixo. O lixo é representado pelo caractere '@'.

---

**Descrição do Registro de Dados.** Deve ser considerada a seguinte organização: campos de tamanho fixo e registros de tamanho fixo. Em adição ao Item 1 da definição formal do índice árvore-B, cada nó (página) da árvore também deve armazenar dois outros campos:

- *folha*: indica se o nó é um nó folha ou não, representado por uma *string* de 1 byte (0 indica que o nó não é folha e 1 indica que o nó é folha);
- *nroChavesIndexadas*: armazena o número de chaves indexadas no nó – tamanho: inteiro de 4 *bytes*.
- *RRNdoNo*: armazena o número do RRN referente ao nó (ou seja, à página de disco) - tamanho: inteiro de 4 *bytes*.

A ordem da árvore-B é 5, ou seja,  $m = 5$ . Portanto, um nó (página) terá 4 chaves de busca e 5 ponteiros. Adicionalmente, considere que deve ser implementada a rotina de *split* durante a inserção. Considere que a distribuição das chaves de busca deve ser o mais uniforme possível, ou seja, considere que a chave de busca a ser promovida deve ser a chave que distribui uniformemente as demais chaves entre o nó à esquerda e o novo nó resultante do particionamento. Considere que a página (ou nó) sendo criada é sempre a página à direita.

**Representação Gráfica de um Nó (Página/Registro de Dados) do índice.** O tamanho de cada registro de dados é de 77 bytes, representado da seguinte forma:

1 byte	4 bytes	4 bytes	4 bytes	4 bytes	8 bytes	4 bytes	4 bytes	8 bytes
<i>folha</i>	<i>nroChaves</i> <i>Indexadas</i>	<i>RRNdo</i> <i>No</i>	$P_1$	$C_1$	$P_{R1}$	$P_2$	$C_2$	$P_{R2}$
0	1...4	5...8	9...12	13...16	17...24	25...28	29...32	33...40

4 bytes	4 bytes	8 bytes	4 bytes	4 bytes	8 bytes	4 bytes
$P_3$	$C_3$	$P_{R3}$	$P_4$	$C_4$	$P_{R4}$	$P_5$
41...44	45...48	49...56	57...60	61...64	65...72	73...76

### Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- Os nomes dos atributos também devem seguir estritamente os nomes definidos na especificação dos mesmos.
- Quando um nó (página) do índice tiver chaves de busca que não forem preenchidas, a chave de busca deve ser representada pelo valor -1. O valor -1 também deve ser usado para denotar ponteiros  $P_i$  ( $1 \leq i \leq m$ ) e  $P_{Ri}$  ( $1 \leq i \leq m - 1$ ) que sejam nulos.

---

## Programa

---

**Descrição Geral.** Implemente um programa em C por meio do qual seja possível realizar a pesquisa e a inserção de dados em arquivos de dados com o auxílio de um índice árvore-B.

**Importante.** A definição da sintaxe de cada comando bem como sua saída devem seguir estritamente as especificações definidas em cada funcionalidade. Para especificar a sintaxe de execução, considere que o programa seja chamado de “programaTrab”. Essas orientações devem ser seguidas uma vez que a correção do funcionamento do programa se dará de forma automática. De forma geral, a primeira entrada da entrada padrão é sempre o identificador de suas funcionalidades, conforme especificado a seguir.

**Descrição Específica.** O programa deve oferecer as seguintes funcionalidades:

Na linguagem SQL, o comando CREATE TABLE é usado para criar uma tabela, a qual é implementada como um arquivo. Geralmente, indica-se um campo (ou um conjunto de campos) que consiste na chave primária da tabela. Isso é realizado especificando-se a cláusula PRIMARY KEY. As funcionalidades [9] e [10] representam exemplos de implementação do índice árvore-B definidos sobre os campos chave primária dos arquivos de dados **veiculo** e **linha**.

Na linguagem SQL, o comando CREATE INDEX é usado para criar um índice sobre um campo (ou um conjunto de campos) que consiste na chave primária de uma tabela já existente. As funcionalidades [9] e [10] representam exemplos de implementação do índice árvore-B definidos sobre os campos chave primária dos arquivos de dados **veiculo** e **linha**.

[9] Crie um arquivo de índice árvore-B para o arquivo de dados **veiculo** definido de acordo com a especificação do primeiro trabalho prático. O campo a ser indexado é **prefixo**. Como o campo **prefixo** foi definido como um campo do tipo *string* no primeiro trabalho prático e o índice árvore-B indexa chaves de busca do tipo inteiro, deve ser utilizada a função `convertePrefixo`, disponibilizada na página do projeto da disciplina, para converter o valor de **prefixo** de *string* para inteiro. Lembre-se que o arquivo de dados **veiculo** pode conter registros logicamente removidos. Entretanto, registros logicamente removidos presentes no arquivo de dados de entrada não devem ter suas chaves de busca correspondentes no arquivo de índice. Lembre-se também que o campo **prefixo** não possui valores repetidos e, como resultado, o índice árvore-B também não deve considerar a possibilidade de valores de chave repetidos. A inserção no arquivo de índice deve ser feita um-a-um. Ou seja, para cada registro não removido presente no arquivo de dados, deve ser feita a inserção de sua chave de busca correspondente no arquivo de índice árvore-B. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Antes de terminar a execução da funcionalidade, deve ser utilizada a função `binarioNaTela`, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo de índice árvore-B.

**Entrada do programa para a funcionalidade [9]:**

```
9 veiculo.bin indicePrefixo.bin
```

**onde:**

- veiculo.bin é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **veiculo** do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.

- indicePrefixo.bin é o arquivo binário de índice árvore-B que indexa o campo **prefixo**. Esse arquivo deve seguir as especificações definidas neste trabalho prático. Antes de serem inseridos no arquivo de índice, os valores do campo **prefixo** devem ser convertidos para valores inteiros usando a função `convertePrefixo`.

**Saída caso o programa seja executado com sucesso:**

Listar o arquivo binário `indicePrefixo.bin` usando a função `binarioNaTela`.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

**Exemplo de execução:**

```
./programaTrab
9 veiculo.bin indicePrefixo.bin
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo indicePrefixo.bin, o
qual indexa o campo prefixo.
```



[10] Crie um arquivo de índice árvore-B para o arquivo de dados **linha** definido de acordo com a especificação do primeiro trabalho prático. O campo a ser indexado é **codLinha**. Lembre-se que o arquivo de dados **linha** pode conter registros logicamente removidos. Entretanto, registros logicamente removidos presentes no arquivo de dados de entrada não devem ter suas chaves de busca correspondentes no arquivo de índice. Lembre-se também que o campo **codLinha** não possui valores repetidos e, como resultado, o índice árvore-B também não deve considerar a possibilidade de valores de chave repetidos. A inserção no arquivo de índice deve ser feita um-a-um. Ou seja, para cada registro não removido presente no arquivo de dados, deve ser feita a inserção de sua chave de busca correspondente no arquivo de índice árvore-B. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Antes de terminar a execução da funcionalidade, deve ser utilizada a função `binarioNaTela`, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo de índice árvore-B.

**Entrada do programa para a funcionalidade [10]:**

10 linha.bin indiceCodLinha.bin

**onde:**

- linha.bin é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **linha** do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.

- indiceCodLinha.bin é o arquivo binário de índice árvore-B que indexa o campo **codLinha**. Esse arquivo deve seguir as especificações definidas neste trabalho prático.

**Saída caso o programa seja executado com sucesso:**

Listar o arquivo binário indiceCodLinha.bin usando a função binarioNaTela.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

**Exemplo de execução:**

```
./programaTrab
```

```
10 linha.bin indiceCodLinha.bin
```

usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo indiceCodLinha.bin, o qual indexa o campo codLinha.

Uma vez criados os índices necessários, eles são automaticamente utilizados pelo SGBD para atender aos demais comandos solicitados, de forma transparente.

[11] Permita a recuperação dos dados de todos os registros do arquivo de dados **veiculo** que satisfaçam um critério de busca determinado pelo usuário sobre o campo **prefixo**, usando o índice árvore-B criado na funcionalidade [9]. Note que somente o campo **prefixo** deve ser utilizado como forma de busca, desde que o índice criado na funcionalidade [9] indexa chaves de busca desse campo. No primeiro trabalho prático, o campo **prefixo** foi definido como do tipo *string*. Portanto, antes de realizar a busca usando o índice árvore-B, deve ser utilizada a função `convertePrefixo`, disponibilizada na página do projeto da disciplina, para converter o valor de **prefixo** de *string* para inteiro. A funcionalidade [11] pode retornar 0 registros (quando nenhum satisfaz ao critério de busca) ou 1 registro (desde que o campo **prefixo** não aceita valores repetidos). Registros marcados como logicamente removidos não devem ser exibidos. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Os dados solicitados devem ser mostrados na saída padrão no mesmo formato definido para a funcionalidade [5].

### **Sintaxe do comando para a funcionalidade [11]:**

```
l1 veiculo.bin indicePrefixo.bin prefixo "valor"
```

#### **onde:**

- **veiculo.bin** é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **veiculo** do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.

- **indicePrefixo.bin** é o arquivo binário de índice árvore-B que indexa o campo **prefixo**. Esse arquivo deve seguir as especificações definidas neste trabalho prático. Antes de serem inseridos no arquivo de índice, os valores do campo **prefixo** devem ser convertidos para valores inteiros usando a função **convertePrefixo**.

- **prefixo** é o campo do arquivo **veiculo** sobre o qual será feita a busca.

- **valor** é o valor do campo **prefixo**.

### **Saída caso o programa seja executado com sucesso:**

Para cada registro, devem ser exibidos seus campos da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por "campo com valor nulo". A data deve ser exibida na forma corrente de escrita. Depois de cada registro, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada no **exemplo de execução**.

### **Mensagem de saída caso não seja encontrado o registro que contém o valor do campo ou o campo pertence a um registro que esteja removido:**

Registro inexistente.

### **Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

### **Exemplo de execução:**

```
./programaTrab  
l1 veiculo.bin indicePrefixo.bin prefixo "BA004"  
Prefixo do veiculo: BA004  
Modelo do veiculo: NEOBUS MEGA  
Categoria do veiculo: ALIMENTADOR  
Data de entrada do veiculo na frota: 29 de maio de 2009  
Quantidade de lugares sentados disponiveis: campo com valor nulo  
--- pule uma linha em branco ---
```

[12] Permita a recuperação dos dados de todos os registros do arquivo de dados **linha** que satisfaçam um critério de busca determinado pelo usuário sobre o campo **codLinha**, usando o índice árvore-B criado na funcionalidade [10]. Note que somente o campo **codLinha** deve ser utilizado como forma de busca, desde que o índice criado na funcionalidade [10] indexa chaves de busca desse campo. A funcionalidade [12] pode retornar 0 registros (quando nenhum satisfaz ao critério de busca) ou 1 registro (desde que o campo **codLinha** não aceita valores repetidos). Registros marcados como logicamente removidos não devem ser exibidos. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Os dados solicitados devem ser mostrados na saída padrão no mesmo formato definido para a funcionalidade [6].

### **Sintaxe do comando para a funcionalidade [12]:**

```
12 linha.bin indiceCodLinha.bin codLinha valor
```

#### **onde:**

- linha.bin é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **linha** do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.
- indiceCodLinha.bin é o arquivo binário de índice árvore-B que indexa o campo **codLinha**. Esse arquivo deve seguir as especificações definidas neste trabalho prático.
- codLinha é o campo do arquivo **linha** sobre o qual será feita a busca.
- valor é o valor do campo **codLinha**.

### **Saída caso o programa seja executado com sucesso:**

Para cada registro, devem ser exibidos seus campos da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por 'campo com valor nulo'. Com relação ao campo aceita cartão, seu valor deve ser exibido como PAGAMENTO SOMENTE COM CARTAO SEM PRESENÇA DE COBRADOR (ao invés de S) ou PAGAMENTO EM CARTAO E DINHEIRO (ao invés de N) ou PAGAMENTO EM CARTAO SOMENTE NO FINAL DE SEMANA (ao invés de F). Depois de cada registro, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada no **exemplo de execução**.

### **Mensagem de saída caso não seja encontrado o registro que contém o valor do campo ou o campo pertence a um registro que esteja removido:**

Registro inexistente.

### **Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

### **Exemplo de execução:**

```
./programaTrab  
12 linha.bin indiceCodLinha.bin codLinha 464  
Nome da linha: campo com valor nulo  
Cor que descreve a linha: AMARELA  
Aceita cartao: PAGAMENTO EM CARTAO SEM COBRADOR  
--- pule uma linha em branco ---
```

[13] Estenda a funcionalidade [7] de forma que, a cada inserção de um registro no arquivo de dados **veiculo**, a chave de busca correspondente a essa inserção seja inserida no arquivo de índice árvore-B. Como o campo **prefixo** foi definido como um campo do tipo *string* no primeiro trabalho prático e o índice árvore-B indexa chaves de busca do tipo inteiro, deve ser utilizada a função `convertePrefixo`, disponibilizada na página do projeto da disciplina, para converter o valor de prefixo de *string* para inteiro antes da chave ser inserida no arquivo de índice árvore-B. Antes de terminar a execução da funcionalidade, deve ser utilizada a função `binarioNaTela`, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo de índice árvore-B.

### Entrada do programa para a funcionalidade [13]:

```
13 veiculo.bin indicePrefixo.bin n
prefixo1 data1 quantidadeLugares1 codLinha1 modelo1 categoria1
prefixo2 data2 quantidadeLugares2 codLinha2 modelo2 categoria2
...
prefixon datan quantidadeLugaresn codLinhan modelon categorian
```

#### onde:

- `veiculo.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **veiculo** do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos. As inserções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.

- `indicePrefixo.bin` é o arquivo binário de índice árvore-B que indexa o campo **prefixo**. Esse arquivo deve seguir as especificações definidas neste trabalho prático. Antes de serem inseridos no arquivo de índice, os valores do campo **prefixo** devem ser convertidos para valores inteiros usando a função `convertePrefixo`.

- `n` é o número de inserções a serem realizadas. Para cada inserção, deve ser informado os valores a serem inseridos no arquivo, para os campos especificados na mesma ordem que a definida nesse trabalho prático, a saber: `prefixo`, `data`, `quantidadeLugares`, `codLinha`, `modelo`, `categoria`. Valores nulos devem ser identificados, na entrada da funcionalidade, por **NULO**. Cada uma das `n` inserções deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre os valores dos campos. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (`"`).

#### Saída caso o programa seja executado com sucesso:

Listar o arquivo binário `indicePrefixo.bin` usando a função `binarioNaTela`.

#### Mensagem de saída caso algum erro seja encontrado:

Falha no processamento do arquivo.

#### Exemplo de execução:

```
./programaTrab
13 veiculo.bin indicePrefixo.bin 2
"AT090" "2019-05-20" 30 333 NULO "VERMELHO"
"XX997" "2015-09-30" 20 672 NULO NULO
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo indicePrefixo.bin.
```



[14] Estenda a funcionalidade [8] de forma que, a cada inserção de um registro no arquivo de dados **linha**, a chave de busca correspondente a essa inserção seja inserida no arquivo de índice árvore-B. Antes de terminar a execução da funcionalidade, deve ser utilizada a função `binarioNaTela`, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo de índice árvore-B.

**Entrada do programa para a funcionalidade [14]:**

```
14 arquivoLinha.bin indiceCodLinha.bin n
codLinha1 aceitaCartao1 nomeLinha1 corLinha1
codLinha2 aceitaCartao2 nomeLinha2 corLinha2
...
codLinhan aceitaCartaon nomeLinhan corLinhan
```

**onde:**

- `linha.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **linha** do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.
- `indiceCodLinha.bin` é o arquivo binário de índice árvore-B que indexa o campo **codLinha**. Esse arquivo deve seguir as especificações definidas neste trabalho prático.
- `n` é o número de inserções a serem realizadas. Para cada inserção, deve ser informado os valores a serem inseridos no arquivo, para os campos especificados na mesma ordem que a definida nesse trabalho prático, a saber: `codLinha`, `aceitaCartao`, `nomeLinha`, `corLinha`. Valores nulos devem ser identificados, na entrada da funcionalidade, por NULO. Cada uma das `n` inserções deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre os valores dos campos. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

**Saída caso o programa seja executado com sucesso:**

Listar o arquivo binário `indiceCodLinha.bin` usando a função `binarioNaTela`.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

**Exemplo de execução:**

```
./programaTrab
14 linha.bin indiceCodLinha.bin 1
333 "S" NULO "VERMELHO"
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo indiceCodLinha.bin.
```

---

## Restrições

---

As seguintes restrições têm que ser garantidas no desenvolvimento do trabalho.

[1] O arquivo de dados deve ser gravado em disco no **modo binário**. O modo texto não pode ser usado.

[2] Os dados do registro descrevem os nomes dos campos, os quais não podem ser alterados. Ademais, todos os campos devem estar presentes na implementação, e nenhum campo adicional pode ser incluído. O tamanho e a ordem de cada campo deve obrigatoriamente seguir a especificação.

[3] Deve haver a manipulação de valores nulos, conforme as instruções definidas.

[4] Não é necessário realizar o tratamento de truncamento de dados.

[5] Devem ser exibidos avisos ou mensagens de erro de acordo com a especificação de cada funcionalidade.

[6] Os dados devem ser obrigatoriamente escritos campo a campo. Ou seja, não é possível escrever os dados registro a registro. Essa restrição refere-se à entrada/saída, ou seja, à forma como os dados são escritos no arquivo.

[7] O(s) aluno(s) que desenvolveu(desenvolveram) o trabalho prático deve(m) constar como comentário no início do código (i.e. NUSP e nome do aluno). Para trabalhos desenvolvidos por mais do que um aluno, não será atribuída nota ao aluno cujos dados não constarem no código fonte.

[8] Todo código fonte deve ser documentado. A **documentação interna** inclui, dentre outros, a documentação de procedimentos, de funções, de variáveis, de partes do

código fonte que realizam tarefas específicas. Ou seja, o código fonte deve ser documentado tanto em nível de rotinas quanto em nível de variáveis e blocos funcionais.

[9] A implementação deve ser realizada usando a linguagem de programação C. As funções das bibliotecas `<stdio.h>` devem ser utilizadas para operações relacionadas à escrita e leitura dos arquivos. A implementação não pode ser feita em qualquer outra linguagem de programação. O programa executará no `[run.codes]`.

---

### Fundamentação Teórica

---

Conceitos e características dos diversos métodos para representar os conceitos de campo e de registro em um arquivo de dados podem ser encontrados nos *slides* de sala de aula e também no livro *File Structures (second edition)*, de Michael J. Folk e Bill Zoellick.

---

### Material para Entregar

---

**Arquivo compactado.** Deve ser preparado um arquivo .zip contendo:

- Código fonte do programa devidamente documentado.
- Makefile para a compilação do programa.
- Um vídeo gravado pelos integrantes do grupo, o qual deve ter, no máximo, 5 minutos de gravação. O vídeo deve explicar o trabalho desenvolvido. Ou seja, o grupo deve apresentar: cada funcionalidade e uma breve descrição de como a funcionalidade foi implementada. Todos os integrantes do grupo devem participar do vídeo, sendo que o tempo de apresentação dos integrantes deve

ser balanceado. Ou seja, o tempo de participação de cada integrante deve ser aproximadamente o mesmo.

**Instruções para fazer o arquivo makefile.** No [run.codes] tem uma orientação para que, no makefile, a diretiva “all” contenha apenas o comando para compilar seu programa e, na diretiva “run”, apenas o comando para executá-lo. Assim, a forma mais simples de se fazer o arquivo makefile é:

all:

```
gcc -o programaTrab *.c
```

run:

```
./programaTrab
```

Lembrando que \*.c já engloba todos os arquivos .c presentes no seu zip. Adicionalmente, no arquivo Makefile é importante se ter um *tab* nos locais colocados acima, senão ele pode não funcionar.

### Instruções de entrega.

O programa deve ser submetido via [run.codes]:

- página: <https://run.codes/Users/login>
- **Turma 1** (segunda-feira): código de matrícula: **7Y6N**
- **Turma 2** (terça-feira): código de matrícula: **H6A3**

O vídeo gravado deve ser submetido por meio de um formulário no qual o grupo vai informar o nome de cada integrante, o número do grupo e um link do Google Drive que contém o vídeo gravado. Não deve ser usado o drive compartilhado da disciplina.

---

### **Critério de Correção**

---

**Critério de avaliação do trabalho.** Na correção do trabalho, serão ponderados os seguintes aspectos.

- Corretude da execução do programa.
- Atendimento às especificações do registro de cabeçalho e dos registros de dados.
- Atendimento às especificações da sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade.
- Qualidade da documentação entregue. A documentação interna terá um peso considerável no trabalho.
- Vídeo. Integrantes que não participarem da apresentação receberão nota 0 no trabalho correspondente.

**Casos de teste no [run.codes].** Juntamente com a especificação do trabalho, serão disponibilizados 70% dos casos de teste no [run.codes], para que os alunos possam avaliar o programa sendo desenvolvido. Os 30% restantes dos casos de teste serão utilizados nas correções.

### **Restrições adicionais sobre o critério de correção.**

- A não execução de um programa devido a erros de compilação implica que a nota final da parte do trabalho será igual a zero (0).
- O não atendimento às especificações do registro de cabeçalho e dos registros de dados implica que haverá uma diminuição expressiva na nota do trabalho.
- O não atendimento às especificações de sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade implica que haverá uma diminuição expressiva na nota do trabalho.

- A ausência da documentação implica que haverá uma diminuição expressiva na nota do trabalho.
- A realização do trabalho prático com alunos de turmas diferentes implica que haverá uma diminuição expressiva na nota do trabalho.
- A inserção de palavras ofensivas nos arquivos e em qualquer outro material entregue implica que a nota final da parte do trabalho será igual a zero (0).
- Em caso de plágio, as notas dos trabalhos envolvidos serão zero (0).

---

### **Data de Entrega do Trabalho**

---

Na data especificada na página da disciplina.

---