**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**
*cdac@icmc.usp.br*

Aluno PAE
**João Paulo Clarindo**
*jpcsantos@usp.br*

Monitores
**Eduardo Souza Rocha**
*eduardos.rocha17@usp.br ou telegram: @edwolt*
**João Francisco Caprioli Barbosa Camargo de Pinho**
*jpinho@usp.br ou telegram: @JotaGHz*
**Maria Júlia Soares De Grandi**
*maju.degrandi@usp.br* ou *telegram: @majudegrandi*

# Primeiro Trabalho Prático

**Este trabalho tem como objetivo realizar diversas operações sobre um arquivo de dados binário. São solicitadas operações de busca, inserção, remoção e atualização. Também é introduzido o uso de índices para auxiliar a busca.**

*O trabalho deve ser feito por 2 alunos da mesma turma. Os alunos devem ser os mesmos que os dos demais trabalhos. Caso haja mudanças, elas devem ser informadas para a docente, o aluno PAE e os monitores. A solução deve ser proposta exclusivamente pelos alunos com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.*

---

## Programa

---

**Descrição Geral**. Implemente um programa em C por meio do qual o usuário possa realizar diversas operações de busca, inserção, remoção e atualização em um arquivo de dados binário, com e sem o uso de índices.

**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.

**Modularização**. É importante modularizar o código. Trechos de programa que aparecerem várias vezes devem ser modularizados em funções e procedimentos.

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

Na linguagem SQL, o comando CREATE INDEX é usado para criar um índice sobre um campo (ou um conjunto de campos) de busca. A funcionalidade [3] representa um exemplo de implementação de um índice linear definido sobre um campo do arquivo de dados.

[3] Crie um arquivo de índice linear sobre um campo de um arquivo de dados. O campo pode ou não possuir valores repetidos. A criação deve ser feita considerando que o arquivo de dados já existe e já possui dados. Valores nulos não devem ser indexados.

Se o campo for do tipo inteiro, o arquivo de índice é definido da seguinte forma:
**Registro de Cabeçalho.** O registro de cabeçalho deve conter o seguinte campo:

- *status*: indica a consistência do arquivo de dados, devido à queda de energia, travamento do programa, etc. Pode assumir os valores '0', para indicar que o arquivo de índice está inconsistente, ou '1', para indicar que o arquivo de índice 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.

**Registros de Dados.** Os registros de dados são de tamanho fixo, com campos de tamanho fixo, definidos da seguinte forma:

- *chaveBusca*: chave de busca referente a um campo do tipo inteiro – inteiro – tamanho: 4 bytes.
- *byteOffset*: número relativo do registro do arquivo de dados referente à *chaveBusca* – inteiro – tamanho: 8 bytes.

Se o campo for do tipo *string*, o arquivo de índice é definido da seguinte forma:

**Registro de Cabeçalho.** O registro de cabeçalho deve conter o seguinte campo:

- *status*: indica a consistência do arquivo de dados, devido à queda de energia, travamento do programa, etc. Pode assumir os valores '0', para indicar que o arquivo de índice está inconsistente, ou '1', para indicar que o arquivo de índice 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.

**Registros de Dados.** Os registros de dados são de tamanho fixo, com campos de tamanho fixo, definidos da seguinte forma:

- *chaveBusca*: chave de busca referente a um campo do tipo *string* – *string* – tamanho: 12 bytes. Se o valor correspondente à chave de busca for maior do que 12 bytes, deve ocorrer o truncamento dos dados. Neste caso, somente os 12 primeiros bytes devem ser considerados.
- *byteOffset*: número relativo do registro do arquivo de dados referente à *chaveBusca* – inteiro – tamanho: 8 bytes.

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 binário de índice.

**Entrada do programa para a funcionalidade [3]:**

```
3 arquivoEntrada.bin campoIndexado tipoDado arquivoIndice.bin
```

**onde:**

- `arquivoEntrada.bin` é o arquivo binário gerado conforme as especificações descritas no trabalho prático da disciplina.
- `campoIndexado` é o nome do atributo utilizado como chave de busca, sendo que o atributo pode ou não possuir valores repetidos.
- `tipoDado` é o tipo de dado do atributo utilizado como chave de busca.
- `arquivoIndice.bin` é um arquivo binário que indexa o arquivo de dados `arquivoDados.bin` e que é gerado conforme as especificações descritas neste trabalho prático.

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

`Listar o arquivo de índice no formato binário usando a função fornecida binarioNaTela.`

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

`Falha no processamento do arquivo.`

**Exemplos de execução:**

```
./programaTrab
3 crime.bin idCrime inteiro idIndice.bin
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo idIndice.bin.`

```
./programaTrab
3 crime.bin marcaCelular string marcaIndice.bin
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo marcaIndice.bin.`

Conforme visto na funcionalidade [2], na linguagem SQL o comando SELECT é usado para listar os dados de uma tabela. Existem várias cláusulas que compõem o comando SELECT. Além das cláusulas SELECT e FROM, outra cláusula muito comum é a cláusula WHERE, que permite que seja definido um critério de busca sobre um ou mais campos, o qual é nomeado como critério de seleção.

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela (ou seja, arquivo que contém os campos)
WHERE critério de seleção (ou seja, critério de busca)

A funcionalidade [3] representa um exemplo de implementação do comando SELECT considerando a cláusula WHERE. A implementação da funcionalidade depende da existência ou não de índices. Caso exista um índice definido sobre o critério de seleção, o índice deve ser utilizado para melhorar o desempenho da consulta (ou seja, para que a consulta seja executada mais rapidamente). Caso contrário, o arquivo deve ser percorrido sequencialmente (busca sequencial).

[4] Permita a recuperação dos dados de todos os registros de um arquivo de dados de entrada, de forma que esses registros satisfaçam um critério de busca determinado pelo usuário. Qualquer campo pode ser utilizado como forma de busca. Adicionalmente, a busca deve ser feita considerando um ou mais campos. Por exemplo, é possível realizar a busca considerando somente o campo *idCrime* ou considerando os campos *lugarCrime* e *marcaCelular*.

Na implementação da funcionalidade, devem ser feitas as seguintes ações:

- Se nenhum índice é definido sobre os campos do critério de busca, então deve ser feita busca sequencial.
- Se o critério de busca for definido em termos de um campo indexado, a busca deve ser realizada usando o índice criado.

Esta funcionalidade pode retornar 0 registros (quando nenhum satisfaz ao critério de busca), 1 registro (quando apenas um satisfaz ao critério de busca), ou vários registros. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas ("). Para a manipulação de *strings* com aspas duplas, pode-se usar a função

scan_quote_string disponibilizada na página do projeto da disciplina. Registros marcados como logicamente removidos não devem ser exibidos.

**Sintaxe do comando para a funcionalidade [4]:**

`4 arquivoEntrada.bin campoIndexado tipoDado arquivoIndice.bin n`
`m1 nomeCampo1 valorCampo1 ... nomeCampom1 valorCampom1`
`campoIndexado2 arquivoIndice2 m2 nomeCampo1 valorCampo1 ... nomeCampom2 valorCampom2`
`...`
`campoIndexadon arquivoIndicen mn nomeCampo1 valorCampo1 ... nomeCampomn valorCampomn`

**onde:**

- `arquivoEntrada.bin` é o arquivo binário gerado conforme as especificações descritas no trabalho prático da disciplina.
- `campoIndexado` é o nome do campo utilizado como chave de busca para a criação do índice linear correspondente.
- `tipoDado` é o tipo de dado do atributo utilizado como chave de busca.
- `arquivoIndice.bin` é o arquivo binário do índice criado sobre o campoIndexado.
- `n` é a quantidade de buscas que devem ser realizadas. Cada busca é especificada em uma linha separada.
- m é a quantidade de pares (`nome do Campo`, `valor do Campo`) que pode ser utilizada como critério em uma busca. Deve ser deixado um espaço em branco entre `campoIndexado` e `arquivoIndice`. Também deve ser deixado um espaço em branco entre `nomeCampo` e `valorCampo` e entre os pares. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

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

`Primeiro, escreva em uma linha “Resposta para a busca 1”. Depois, o valor de cada campo de cada registro deve ser mostrado em uma única linha, e deve ser separado por vírgula. Caso o campo seja nulo, deve ser exibido: NULO. Ordem de exibição dos campos: idCrime, dataCrime, numeroArtigo, lugarCrime, descriçãoCrime, marcaCelular. Depois, escreva em uma nova linha “Resposta para a busca 2” e, na sequência, o valor de cada campo de cada registro deve ser mostrado em uma única linha, e deve ser separado por vírgula. Caso o campo seja nulo, deve ser exibido: NULO. Ordem de exibição dos campos: idCrime, dataCrime,`

numeroArtigo, lugarCrime, descriçãoCrime, marcaCelular. Esse padrão de saída deve ser seguido até que o valor de n seja alcançado. Caso nenhum registro seja retornado na busca, deve ser exibida a mensagem Registro inexistente. Veja um exemplo 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
4 crime.bin idCrime inteiro idIndice.bin 4
1 idCrime 1
1 lugarCrime "COLINA"
2 dataOcorrencia "28/02/2019" numeroArtigo 171
1 lugarCrime "SUMARE"
Resposta para a busca 1
1, 1, 08/04/2017, 157, SAO CARLOS, ROUBO, NOKIA
Resposta para a busca 2
Registro inexistente.
Resposta para a busca 3
43, 28/02/2019, 171, RIO DE JANEIRO, ROUBO, NULO
68, 28/02/2019, 171, CURITIBA, ROUBO, MOTOROLA
Resposta para a busca 4
Registro inexistente.
```

Na linguagem SQL, o comando DELETE é usado para remover dados em uma tabela. Para tanto, devem ser especificados quais dados (ou seja, registros) devem ser removidos, de acordo com algum critério.

DELETE FROM tabela (ou seja, arquivo que contém os campos)

WHERE critério de seleção (ou seja, critério de busca)

A funcionalidade [5] representa um exemplo de implementação do comando DELETE.

[5] Permita a remoção lógica de registros, baseado na *abordagem estática* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve seguir estritamente a matéria apresentada em sala de aula. Os registros a serem removidos devem ser aqueles que satisfaçam um critério de busca (ou busca combinada) determinado pelo usuário, conforme detalhado na funcionalidade [4]. A funcionalidade [5] deve ser executada *n* vezes seguidas. Em situações nas quais um determinado critério de busca não seja satisfeito, ou seja, caso a solicitação do usuário não retorne nenhum registro a ser removido, o programa deve continuar a executar as remoções até completar as *n* vezes seguidas.

No arquivo de dados, a marcação dos registros logicamente removidos deve ser feita armazenando-se 1 no primeiro campo do registro, que é um campo do tipo inteiro. Os demais caracteres devem permanecer como estavam anteriormente. A remoção de um registro no arquivo de dados indica que a entrada correspondente no arquivo de índice também deve ser removida. No arquivo de índice, a entrada deve ser completamente removida.

Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar as saídas dos arquivos binários.

**Sintaxe do comando para a funcionalidade [5]:**

`5 arquivoEntrada.bin campoIndexado tipoDado arquivoIndice.bin n`

$m_1$ `nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{m1}$ `valorCampo`$_{m1}$

$m_2$ `nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{m2}$ `valorCampo`$_{m2}$

`...`

$m_n$ `nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{mn}$ `valorCampo`$_{mn}$

**onde:**

- `arquivoEntrada.bin` é o arquivo binário gerado conforme as especificações descritas no trabalho prático da disciplina.

- `campoIndexado` é o nome do campo utilizado como chave de busca para a criação do índice linear correspondente.

- `tipoDado` é o tipo de dado do atributo utilizado como chave de busca.

- `arquivoIndice.bin` é o arquivo binário do índice criado sobre o campoIndexado.

- `n` é a quantidade de remoções que devem ser realizadas. Cada remoção é especificada em uma linha separada.

- `m` é a quantidade de pares (`nome do Campo`, `valor do Campo`) que pode ser utilizada como critério em uma remoção. Deve ser deixado um espaço em branco entre `campoIndexado` e `arquivoIndice`. Também deve ser deixado um espaço em branco entre `nomeCampo` e `valorCampo` e entre os pares. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

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

`Listar os arquivos de dados e de índice no formato binário usando a função fornecida binarioNaTela.`

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

`Falha no processamento do arquivo.`

**Exemplo de execução:**

```
./programaTrab
5 crime.bin idCrime inteiro idIndice.bin 3
1 idCrime 1
1 lugarCrime "COLINA"
2 dataOcorrencia "28/02/2019" numeroArtigo 171
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos crime.bin e idIndice.bin, os quais foram atualizados com as remoções.`

Na linguagem SQL, o comando INSERT INTO é usado para inserir dados em uma tabela. Para tanto, devem ser especificados os valores a serem armazenados em cada coluna da tabela, de acordo com o tipo de dado definido. A funcionalidade [6] representa um exemplo de implementação do comando INSERT INTO.

[6] Permita a inserção de registros adicionais, baseado na *abordagem estática* de registros logicamente removidos. A implementação dessa funcionalidade deve seguir estritamente a matéria apresentada em sala de aula. Não é necessário realizar o tratamento de truncamento de dados. Campos com valores nulos, na entrada da funcionalidade, devem ser identificados com NULO. A funcionalidade [6] deve ser executada *n* vezes seguidas. A inserção de um registro no arquivo de dados indica que a entrada correspondente deve ser inserida no arquivo de índice. A implementação da inserção da entrada no arquivo de índice deve seguir estritamente a matéria apresentada 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 as saídas dos arquivos binários.

**Sintaxe do comando para a funcionalidade [6]:**

`6 arquivoEntrada.bin campoIndexado tipoDado arquivoIndice.bin n`
`idCrime`$_1$ `dataCrime`$_1$ `numeroArtigo`$_1$ `lugarCrime`$_1$ `descricaoCrime`$_1$ `marcaCelular`$_1$
`idCrime`$_2$ `dataCrime`$_2$ `numeroArtigo`$_2$ `lugarCrime`$_2$ `descricaoCrime`$_2$ `marcaCelular`$_2$
`...`
`idCrime`$_n$ `dataCrime`$_n$ `numeroArtigo`$_n$ `lugarCrime`$_n$ `descricaoCrime`$_n$ `marcaCelular`$_n$

**onde:**

- `arquivoEntrada.bin` é o arquivo binário gerado conforme as especificações descritas no trabalho prático da disciplina.
- `campoIndexado` é o nome do campo utilizado como chave de busca para a criação do índice linear correspondente.
- `tipoDado` é o tipo de dado do atributo utilizado como chave de busca.
- `arquivoIndice.bin` é o arquivo binário do índice criado sobre o campoIndexado.
- `n` é a quantidade de inserções a serem realizadas. Cada inserção é especificada em uma linha separada.

- `idCrime`, `dataCrime`, `numeroArtigo`, `lugarCrime`, `descricaoCrime`, `marcaCelular` são os valores a serem inseridos no arquivo, para os campos especificados na mesma ordem que a definida nesse trabalho prático. Não existe truncamento de dados. 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 os arquivos de dados e de índice saída no formato binário usando a função fornecida binarioNaTela.`

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

`Falha no processamento do arquivo.`

**Exemplo de execução:**

```
./programaTrab
6 crime.bin lugarCrime string lugarIndice.bin 2
1414 "03/01/2021" 157 "SAO BERNARDO DO CAMPO" "ROUBO" NULO
691 31/08/2019 NULO NULO "ROUBO (ART. 157) - TRANSEUNTE" "SAMSUNG"
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos crime.bin e lugarIndice.bin, os quais foram atualizados com as inserções.`

Na linguagem SQL, o comando UPDATE é usado para atualizar dados em uma tabela. Para tanto, devem ser especificados quais valores de dados de quais campos devem ser atualizados, de acordo com algum critério de busca dos registros a serem atualizados.

UPDATE tabela (ou seja, arquivo que contém os dados)

SET quais colunas e quais valores (ou seja, quais campos e seus valores)

WHERE critério de seleção (ou seja, critério de busca)

A funcionalidade [7] representa um exemplo de implementação do comando UPDATE.

[7] Permita a atualização de registros. Os registros a serem atualizados devem ser aqueles que satisfaçam um critério de busca determinado pelo usuário, conforme especificado na funcionalidade [4]. O campo utilizado como busca não precisa ser, necessariamente, o campo a ser atualizado. Por exemplo, pode-se buscar pelo campo *idCrime*, e pode-se atualizar o campo *dataCrime*. Quando o tamanho do registro atualizado for maior do que o tamanho do registro atual, então o registro atual deve ser logicamente removido (funcionalidade [5]) e o registro atualizado deve ser inserido como um novo registro (funcionalidade [6]). Quando o tamanho do registro atualizado for menor ou igual do que o tamanho do registro atual, então a atualização deve ser feita diretamente no registro existente, sem a necessidade de remoção e posterior inserção. Neste caso, o lixo que porventura permaneça no registro atualizado deve ser identificado pelo caractere '$'. Campos a serem atualizados com valores nulos devem ser identificados, na entrada da funcionalidade, com NULO. A funcionalidade [6] deve ser executada *n* vezes seguidas. Em situações nas quais um determinado critério de busca não seja satisfeito, ou seja, caso a solicitação do usuário não retorne nenhum registro a ser atualizado, o programa deve continuar a executar as atualizações até completar as *n* vezes seguidas. A atualização de um registro no arquivo de dados indica que a entrada correspondente deve ser atualizada no arquivo de índice. A implementação da atualização do arquivo de índice deve seguir estritamente a matéria apresentada 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 as saídas dos arquivos binários.

**Entrada do programa para a funcionalidade [7]:**

`7 arquivoEntrada.bin campoIndexado tipoDado arquivoIndice.bin n`

$m_1$ `nomeCampoBusca`$_1$ `valorCampoBusca`$_1$ `...` `nomeCampoBusca`$_{m1}$ `valorCampoBusca`$_{m1}$ $p_1$ `nomeCampoAtualiza`$_1$ `valorCampoAtualiza`$_1$ `...` `nomeCampoAtualiza`$_{p1}$ `valorCampoAtualiza`$_{p1}$

$m_2$ `nomeCampoBusca`$_1$ `valorCampoBusca`$_1$ `...` `nomeCampoBusca`$_{m2}$ `valorCampoBusca`$_{m2}$ $p_2$ `nomeCampoAtualiza`$_1$ `valorCampoAtualiza`$_1$ `...` `nomeCampoAtualiza`$_{p2}$ `valorCampoAtualiza`$_{p2}$
`...`
$m_n$ `nomeCampoBusca`$_1$ `valorCampoBusca`$_1$ `...` `nomeCampoBusca`$_{mn}$ `valorCampoBusca`$_{mn}$ $p_n$ `nomeCampoAtualiza`$_1$ `valorCampoAtualiza`$_1$ `...` `nomeCampoAtualiza`$_{pn}$ `valorCampoAtualiza`$_{pn}$

**onde:**

- `arquivoEntrada.bin` é o arquivo binário gerado conforme as especificações descritas no trabalho prático da disciplina.
- `campoIndexado` é o nome do campo utilizado como chave de busca para a criação do índice linear correspondente.
- `tipoDado` é o tipo de dado do atributo utilizado como chave de busca.
- `arquivoIndice.bin` é o arquivo binário do índice criado sobre o campoIndexado.
- `n` é o número de atualizações a serem realizadas. Cada atualização é especificada em uma linha separada.
- m é a quantidade de pares (`nomeCampoBusca, valorCampoBusca`) que pode ser utilizada como critério em uma busca. Deve ser deixado um espaço em branco entre `nomeCampoBusca` e `valorCampoBusca` e entre os pares. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").
- p é a quantidade de pares (`nomeCampoAtualiza, valorCampoAtualiza`) que devem ser atualizados. Deve ser deixado um espaço em branco entre `nomeCampoBusca` e `valorCampoBusca` e entre os pares. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas ("). Valores nulos devem ser identificados, na entrada da funcionalidade, por NULO.

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

`Listar os arquivos de saída no formato binário usando a função fornecida binarioNaTela.`

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

```
Falha no processamento do arquivo.
```

**Exemplo de execução:**

```
./programaTrab
7 crime.bin lugarCrime string lugarIndice.bin 2
1 idCrime 1414 1 lugarCrime "ARARAQUARA"
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída dos arquivos crime.bin e
lugarIndice.bin, os quais foram atualizados com as inserções.
```

## 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. O uso da webcam é obrigatório.

**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. Adicionalmente, para utilizar a função binarioNaTela, é necessário usar a flag -lmd. Assim, a forma mais simples de se fazer o arquivo makefile é:

```
all:
	gcc -o programaTrab *.c -lmd
run:
	./programaTrab
```

Lembrando que *.c já engloba todos os arquivos .c presentes no arquivo 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: **NJMU**
- **Turma 2** (terça-feira): código de matrícula: **9NMB**

O vídeo gravado deve ser submetido por meio da página da disciplina no e-disciplinas, no qual o grupo vai informar o nome de cada integrante, o número do grupo e um link que contém o vídeo gravado. Ao submeter o link, verifique se o mesmo pode ser acessado. Vídeos cujos links não puderem ser acessados receberão nota zero.

## 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).

**Bom Trabalho!**