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*

Segundo Trabalho Prático

Este trabalho tem como objetivo indexar arquivos de dados usando um índice árvore-B* e utilizar esse índice para melhorar funcionalidades de busca (pesquisa) e junção.

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.

Descrição de páginas de disco da árvore-B*

No trabalho é usado o conceito de páginas de disco, desde que cada nó da árvore-B* é do tamanho de uma página de disco. Cada página de disco tem o tamanho fixo de 76 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: $<P_1, <C_1,P_{R1}>, P_2, <C_2,P_{R2}>, ..., P_{q-1}, <C_{q-1},P_{Rq-1}>, P_q>$, 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)
   - $C_1 < C_2 < ... < C_{q-1}$.
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 (2m-1)/3 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 $\lfloor(2m-1)/3\rfloor$ 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 árvore-B*, 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*.
- *nroNiveis:* armazena o número de níveis do índice árvore-B*. Inicialmente, a árvore-B* está vazia e, portanto, *nroNiveis* = 0. Quando as primeiras *m* chaves de busca são inseridas, o nó raiz é igual ao nó folha e, nesse caso, *nroNiveis* = 1. Quando ocorrer o primeiro *split*, *nroNiveis* = 2. Depois disso, cada vez que o nível da árvore aumentar, *nroNiveis* deve ser incrementado – tamanho: inteiro de 4 *bytes*.
- *nroChaves:* armazena o número de chaves de busca indexadas no índice árvore-B*. Inicialmente, a árvore-B* está vazia e, portanto, *nroChaves* = 0. A cada chave de busca inserida na árvore-B*, *nroChaves* é incrementado – tamanho: inteiro de 4 *bytes*.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho é de 76 bytes, representado da seguinte forma:

1 byte	4 bytes				4 bytes				4 bytes				4 bytes				59 bytes	
status	*noRaiz*				*proxRRN*				*nroNiveis*				*nroChaves*				*lixo (caractere '$')*	
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*...*	*75*

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 é 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 é seguido neste trabalho por simplificação.
- Os 59 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:

- *nível*, indicando o nível no qual o nó se encontra. Quando um nó é um nó-folha, *nível* = 1. Quando um nó aponta para um nó-folha, *nível* = 2. Quando um nó aponta para um outro nó que, por sua vez, aponta para um nó-folha, *nível* = 3. E assim sucessivamente – tamanho: inteiro de 4 *bytes*
- *n*, indicando o número de chaves presentes no nó – tamanho: inteiro de 4 *bytes*.

A ordem da árvore-B* é 5, ou seja, $m = 5$. Portanto, um nó (página) terá 4 chaves e 5 descendentes. Lembrando que, em aplicações reais, a ordem da árvore-B* é muito maior, para acomodar mais chaves. A proposta da árvore-B* é que ela seja larga e baixa, para diminuir o número de acessos a disco.

Representação Gráfica de um Nó (Página/Registro de Dados) do índice. O tamanho do registro de cabeçalho é de 76 bytes, representado da seguinte forma:

4 bytes	4 bytes	4 bytes	4 bytes	8 bytes	4 bytes	4 bytes	8 bytes
nível	*n*	P_1	C_1	P_{R1}	P_2	C_2	P_{R2}
0...3	4...7	8...11	12...15	16...23	24...27	28...31	32...39

4 bytes	4 bytes	8 bytes	4 bytes	4 bytes	8 bytes	4 bytes
P_3	*C3*	P_{R3}	P_4	C_4	P_{R4}	P_5
40...43	44...47	48...55	56...59	60...63	64...71	72...75

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 e o ponteiro para o arquivo de dados deve ser representado pelo valor -1.
- O valor -1 deve ser usado para denotar que um ponteiro P_i $(1 \leq i \leq m)$ de um nó da árvore-B* é nulo.

Detalhes sobre o algoritmo de inserção. Considere que:

- Devem ser implementadas as rotinas *split 1-to-2* e *split 2-to-3* durante a inserção.
- O nó raiz tem o mesmo tamanho dos demais nós da árvore-B*. Assim, quando o *split* ocorrer no nó raiz, deve ser utilizada a rotina *split 1-to-2*. Quando o *split* ocorrer em qualquer outro nó, deve ser utilizada a rotina *split 2-to-3*.
- A página criada nas rotinas *split 1-to-2* e *split 2-to-3* é sempre a página à direita.
- A distribuição das chaves de busca deve ser o mais uniforme possível, ou seja, considere que a(s) chave(s) de busca a ser(em) promovida(s) deve(m) ser a(s)

chave(s) que distribui(em) o mais uniformemente possível as demais chaves entre os nós. Quando necessário, o nó mais à direita deverá conter uma chave de busca a menos.

- Deve ser implementada a rotina de redistribuição durante a inserção. Primeiramente, deve-se tentar realizar a redistribuição com a página irmã à esquerda. Caso isso não seja possível, deve-se tentar realizar a redistribuição com a página irmã à direita. Caso isso não seja possível, deve ser utilizada a rotina split 1-to-2 (nó raiz) ou split 2-to-3 (demais nós) com a página irmã à direita. Caso não haja página irmã à direita, a rotina split 1-to-2 (nó raiz) ou split 2-to-3 (demais nós) deve ser realizada com a página irmã à esquerda.

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.

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 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. A funcionalidade [8] representa um exemplo de implementação de um índice árvore-B* definido sobre o campo chave primária de um arquivo de dados.

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 [8] representa um exemplo de implementação de um índice árvore-B* definido sobre o campo chave primária de um arquivo de dados.

[8] Crie um arquivo de índice árvore-B* para um arquivo de dados de entrada já existente, que é o arquivo de dados definido de acordo com a especificação dos trabalhos práticos, e que pode conter registros logicamente removidos. O campo a ser indexado, ou seja, a chave de busca, é o campo *idCrime*. Esse campo não possui repetição. Registros logicamente removidos presentes no arquivo de dados de entrada não devem ter suas chaves de busca correspondentes no arquivo de índice. 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*.

Sintaxe do comando para a funcionalidade [8]:

```
8 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 não possuir valores repetidos.
- `tipoDado` é o tipo de dado do atributo utilizado como chave de busca.
- `arquivoIndice.bin` é um arquivo binário do índice árvore-B que indexa o arquivo de dados de entrada 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.`

Exemplo de execução:

```
./programaTrab
2 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.`

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 [9] 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).

[9] 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, ou seja, o índice árvore-B*.

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 (").

A manipulação do arquivo de índice árvore-B* deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. 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 [9]:

`9 arquivoEntrada.bin campoIndexado tipoDado arquivoIndice.bin n`
m_1 `nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{m1}$ `valorCampo`$_{m1}$
`campoIndexado`$_2$ `arquivoIndice`$_2$ m_2 `nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{m2}$ `valorCampo`$_{m2}$
`...`
`campoIndexado`$_n$ `arquivoIndice`$_n$ 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 árvore-B* correspondente.
- `tipoDado` é o tipo de dado do atributo utilizado como chave de busca.
- `arquivoIndice.bin` é o arquivo binário do índice árvore-B 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
9 crime.bin idCrime inteiro idIndice.bin 4
1 idCrime 1
1 lugarCrime "COLINA"
2 dataOcorrencia "28/02/2019" numeroArtigo 171
1 idCrime 1 lugarCrime "SUMARE"
Resposta para a busca 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 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 [10] representa um exemplo de implementação do comando INSERT INTO.

[10] 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 [10] 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. 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 (de dados e de índice).

Sintaxe do comando para a funcionalidade [10]:

```
10 arquivoEntrada.bin campoIndexado tipoDado arquivoIndice.bin n
idCrime1 dataCrime1 numeroArtigo1 lugarCrime1 descricaoCrime1 marcaCelular1
idCrime2 dataCrime2 numeroArtigo2 lugarCrime2 descricaoCrime2 marcaCelular2
...
idCrimen dataCrimen numeroArtigon lugarCrimen descricaoCriminen marcaCelularn
```

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 árvore-B 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
10 crime.bin idCrime inteiro idIndice.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 idIndice.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!