Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Estrutura de Dados III (SCC0607)
docente
Profa. Dra. Cristina Dutra de Aguiar Ciferri (cdac@icmc.usp.br)
alunos PAE
João Paulo Clarindo (jpcsantos@usp.br)
monitores
Helbert Moreira Pinto [helbert.moreira@usp.br] telegram: @HelbertMP
Matheus Carvalho Raimundo [mcarvalhor@usp.br] telegram: @mcarvalhor

Primeiro Trabalho Prático

Este trabalho tem como objetivo armazenar dados em arquivos binários de acordo com uma organização de campos e registros, bem como recuperar os dados armazenados com e sem o uso de um índice primário.

De acordo com o critério de avaliação da disciplina, o trabalho deve ser feito por, no máximo, 2 alunos da mesma turma. 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.

Descrição dos arquivos de dados
* Fundamentos da disciplina de Bases de Dados *

O trabalho tem como objetivo armazenar e recuperar dados relacionados a pessoas que seguem pessoas no **twitter**. Nesse sentido, a disciplina de Estrutura de Dados III é uma disciplina fundamental para a disciplina de Bases de Dados. Na disciplina de Bases de Dados, é ensinado que o projeto deve ser feito em dois arquivos. O primeiro arquivo é relacionado às pessoas, armazenando apenas dados relacionados a essas pessoas. O segundo arquivo é um arquivo de relacionamento, que relaciona pessoas que seguem outras pessoas. Visando atender aos requisitos de um bom projeto do banco de dados, são definidos dois arquivos de dados a serem utilizados nos trabalhos práticos: arquivo de dados **pessoa** e arquivo de dados **segue**. Neste primeiro trabalho prático, é implementado o arquivo **pessoa**.

Descrição do arquivo de dados pessoa

O arquivo de dados **pessoa** possui um registro de cabeçalho e 0 ou mais registros de dados, conforme a definição a seguir.

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

- *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 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.
- *quantidadePessoas*: armazena a quantidade de pessoas (ou o número de registros) presentes no arquivo – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 64 bytes, representado da seguinte forma:

<table>
<tr><td>1 byte</td><td colspan="4">4 bytes</td><td colspan="4">59 bytes</td></tr>
<tr><td>status</td><td colspan="4">quantidade Pessoas</td><td colspan="4">lixo (caractere '$')</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>...</td><td>61</td><td>62</td><td>63</td></tr>
</table>

Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Para caber em páginas de disco (que são definidas em potência de 2), o registro de cabeçalho foi definido como uma potência de 2, no mesmo tamanho dos registros de dados). Portanto, o registro de cabeçalho tem o tamanho de 64 bytes, sendo 5 bytes preenchidos com dados necessários para o desenvolvimento do trabalho, e os bytes restantes preenchidos com lixo. O lixo é representado pelo caractere '$'.

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

- *removido*: indica se o registro está logicamente removido. Pode assumir os valores '0', para indicar que o registro está marcado como removido, ou '1', para indicar que o registro não está marcado como removido. Ao se inserir um novo registro, o valor de *removido* deve ser '1'– tamanho: *string* de 1 byte.
- *idPessoa*: código sequencial que identifica univocamente cada registro do arquivo de dados – inteiro – tamanho: 4 bytes.
- *nomePessoa*: nome completo da pessoa – *string* – tamanho: 40 bytes.
- *idadePessoa*: idade da pessoa – inteiro – tamanho: 4 bytes.
- *twitterPessoa*: twitter da Pessoa – *string* – tamanho: 15 bytes.

Os dados são fornecidos juntamente com a especificação deste trabalho prático por meio de um arquivo .csv, sendo que sua especificação se encontra disponível na página da disciplina. No arquivo .csv, o separador de campos é vírgula (,).

Representação Gráfica dos Registros de Dados. O tamanho de cada registro de dados deve ser de 64 bytes, representado da seguinte forma:

<table>
<tr><td>1 byte</td><td colspan="4">4 bytes</td><td colspan="2">40 bytes</td><td colspan="2">4 bytes</td><td colspan="4">15 bytes</td></tr>
<tr><td>removido</td><td colspan="4">idPessoa</td><td colspan="2">nomePessoa</td><td colspan="2">idadePessoa</td><td colspan="4">twitterPessoa</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td colspan="6">...</td><td>60</td><td>61</td><td>62</td><td>63</td></tr>
</table>

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Quando necessário, as *strings* devem ser finalizadas com '\0' e o lixo deve ser identificado pelo caractere '$'. Nenhum *byte* do registro deve permanecer vazio, ou seja, cada *byte* deve armazenar um valor válido ou '\0' ou '$'.

- Os campos *idPessoa* e *twitterPessoa* não aceitam valores nulos. Os demais campos aceitam valores nulos. O arquivo .csv com os dados de entrada já garante essa característica.
- Os valores nulos devem ser representados da seguinte forma:
    - se o campo é inteiro ou de dupla precisão, então armazena-se o valor -1
    - se o campo é do tipo *string*, então armazena-se '\0$$$$$$$$$$'
- Quando necessário, deve ser realizado o tratamento de truncamento de dados. Nesses casos, devem ser considerados os *bytes* iniciais do dado sendo informado. Os *bytes* finais que não couberem no tamanho do campo são descartados.
- Para caber em páginas de disco (que são definidas em potência de 2), os registros de dados foram definidos como uma potência de 2. Ou seja, cada registro de dados tem tamanho de 64 bytes.

Descrição do Arquivo de Índice Primário

O arquivo de dados **pessoa** é indexado por um índice primário, que é definido sobre o campo *idPessoa*. O índice primário também é um arquivo, sendo chamado de arquivo de índice **indexaPessoa**. A especificação de **indexaPessoa** é feita a seguir.

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

- *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 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.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 8 bytes, representado da seguinte forma:

1 byte	7 bytes						
status	*lixo (caractere '$')*						
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*

Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Para caber em páginas de disco (que são definidas em potência de 2), o registro de cabeçalho foi definido como uma potência de 2, no mesmo tamanho dos registros de dados). Portanto, o registro de cabeçalho tem o tamanho de 8 bytes, sendo *1 byte* preenchido com dados necessários para o desenvolvimento do trabalho, e os bytes restantes preenchidos com lixo. O lixo é representado pelo caractere '$'.

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

- *idPessoa*: código sequencial que identifica univocamente cada registro de pessoa armazenado no arquivo de dados pessoa: *inteiro – tamanho: 4 bytes.*
- *RRN*: número relativo do registro. Contém o RRN do registro do arquivo de dados que se refere ao *idPessoa – inteiro – tamanho: 4 bytes*.

Representação Gráfica dos Registros de Dados. O tamanho de cada registro de dados deve ser de 8 bytes, representado da seguinte forma:

4 bytes				4 bytes			
idPessoa				*RRN*			
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.

- Os campos *idPessoa* e *RRN* não aceitam valores nulos.
- Os valores de *idPessoa* devem ser ordenados de forma crescente.
- Para caber em páginas de disco (que são definidas em potência de 2), os registros de dados foram definidos como uma potência de 2. Ou seja, cada registro de dados tem tamanho de 8 bytes.

Programa

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 Geral. Implemente um programa em C que ofereça as seguintes funcionalidades.

[1] Permita a leitura de vários registros obtidos a partir de um arquivo de entrada (arquivo no formato .csv) e a gravação desses registros em um arquivo de dados de saída. O arquivo de entrada é fornecido juntamente com a especificação do projeto, enquanto o arquivo de dados de saída (arquivo **pessoa**) deve ser gerado como parte deste trabalho prático. Nessa funcionalidade, ocorrem várias inserções no arquivo **pessoa**. O arquivo de índice **indexaPessoa** já teve estar ativado. Assim, para cada inserção de um registro no arquivo de dados **pessoa**, deve ser inserido também uma entrada correspondente no arquivo de índice **indexaPessoa**. 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. Lembre-se de manipular o campo "status" do registro de cabeçalho adequadamente. Primeiro ele deve começar com o valor '0' (arquivo inconsistente) e, só ao final da execução do programa e ao final de todas as gravações, ele deve mudar para o valor '1' (arquivo consistente).

Entrada do programa para a funcionalidade [1]:

`1 arquivoEntrada.csv arquivoPessoa.bin arquivoIndexaPessoa.bin`

onde:

- `arquivoEntrada.csv` é um arquivo .csv que contém os valores dos campos dos registros a serem armazenados no arquivo de dados **pessoa**.
- `arquivoPessoa.bin` é o arquivo de dados **pessoa** no formato binário gerado conforme as especificações descritas neste trabalho prático.
- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** no formato binário gerado conforme as especificações descritas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

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

Mensagem de saída caso algum erro seja encontrado:

`Falha no carregamento do arquivo.`

Exemplo de execução:

```
./programaTrab
1 arquivoEntrada.csv arquivoPessoa.bin arquivoIndexaPessoa.bin
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos binários arquivoPessoa e arquivoIndexaPessoa.bin
```

[2] Permita a recuperação dos dados, de todos os registros, armazenados no arquivo de dados **pessoa**, mostrando os dados de forma organizada na saída padrão para permitir a distinção dos campos e registros. O tratamento de 'lixo' deve ser feito de forma a permitir a exibição apropriada dos dados. Registros marcados como logicamente removidos não devem ser exibidos.

Entrada do programa para a funcionalidade [2]:

```
2 arquivoPessoa.bin
```

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** no formato binário gerado conforme as especificações descritas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

```
A saída deve ser exibida da seguinte forma. Para cada registro:
"Dados da pessoa de código " escrever o valor de idPessoa
"Nome: " escrever o valor de nomePessoa
"Idade: " escrever o valor de idadePessoa "anos"
"Twitter: " escrever o valor de twitterPessoa
Valores nulos devem substituídos pelo caractere "-".
Deve ser deixada uma linha em branco entre cada registro listado.
```

Mensagem de saída caso não existam registros:

```
Registro inexistente.
```

Mensagem de saída caso algum erro seja encontrado:

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

Exemplo de execução:

```
./programaTrab
2 arquivoPessoa.bin
Dados da pessoa de código 25
Nome: Samantha Pereira Santos
Idade: 13 anos
Twitter: samanthaps
linha em branco
Dados da pessoa de código 45
Nome: Vitória Prado Campos
Idade: -
Twitter: vivica
linha em branco
...
```

[3] Permita a busca dos dados de todos os registros do arquivo **pessoa** que satisfaçam um critério de busca determinado pelo usuário. Por exemplo, o usuário pode solicitar a exibição do registro referente a *idPessoa* ou de todos os registros de uma determinada *idade*. Qualquer campo pode ser utilizado como forma de busca. Porém, a busca não deve ser combinada. Ou seja, apenas um único campo deve ser utilizado como critério de busca. Em situações nas quais a busca for feita considerando o campo *idPessoa*, deve ser utilizado o arquivo de índice **indexaPessoa** para se fazer a busca. Tem-se, nesse caso, uma busca indexada. Para os demais casos, deve ser feita uma busca sequencial. Registros marcados como logicamente removidos não devem ser exibidos. Os dados solicitados devem ser mostrados no mesmo formato definido para a funcionalidade [2].

Sintaxe do comando para a funcionalidade [3]:

```
3 arquivoPessoa.bin arquivoIndexaPessoa.bin NomeDoCampo valor
```

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** gerado conforme as especificações descritas neste trabalho prático.

- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** gerado conforme as especificações descritas neste trabalho prático.

- `nomeDoCampo` é o nome do campo que está sendo usado na busca.

- `valor` é o valor para o campo que está sendo usado na busca.

Saída caso o programa seja executado com sucesso:

```
Podem ser encontrados vários registros que satisfaçam à condição de busca. A saída deve ser exibida da seguinte forma. Para cada registro:
"Dados da pessoa de código " escrever o valor de idPessoa
"Nome: " escrever o valor de nomePessoa
"Idade: " escrever o valor de idadePessoa "anos"
"Twitter: " escrever o valor de twitterPessoa
Valores nulos devem substituídos pelo caractere "-".
Deve ser deixada uma linha em branco entre cada registro listado.
```

Mensagem de saída caso não existam registros:

```
Registro inexistente.
```

Mensagem de saída caso algum erro seja encontrado:

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

Exemplo de execução (são mostrados alguns registros ilustrativos)**:**

```
./programaTrab
3 arquivoPessoa.bin arquivoIndexaPessoa.bin idPessoa 25
Dados da pessoa de código 25
Nome: Samantha Pereira Santos
Idade: 13 anos
Twitter: samanthaps
linha em branco
```

[4] 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. Quando necessário, deve ser realizado o tratamento de truncamento de dados. Nesses casos, devem ser considerados os *bytes* iniciais do dado sendo informado. Os *bytes* finais que não couberem no tamanho do campo são descartados. Campos com valores nulos, na entrada da funcionalidade, devem ser identificados com NULO. Para cada inserção de um registro no arquivo de dados **pessoa**, deve ser inserido também uma entrada correspondente no arquivo de índice **indexaPessoa**. A funcionalidade [4] deve ser executada *n* vezes seguidas. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída dos arquivos binários. Lembre-se de manipular o campo "status" do registro de cabeçalho adequadamente. Primeiro ele deve começar com o valor '0' (arquivo inconsistente) e, só ao final da execução do programa e ao final de todas as gravações, ele deve mudar para o valor '1' (arquivo consistente).

Entrada do programa para a funcionalidade [4]:

`4 arquivoPessoa.bin arquivoIndexaPessoa.bin n`

`idPessoa`$_1$ `nomePessoa`$_1$ `idadePessoa`$_1$ `twitterPessoa`$_1$

`idPessoa`$_2$ `nomePessoa`$_2$ `idadePessoa`$_2$ `twitterPessoa`$_2$

`...`

`idPessoa`$_n$ `nomePessoa`$_n$ `idadePessoa`$_n$ `twitterPessoa`$_n$

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** no formato binário que segue as especificações definidas neste trabalho prático.
- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** no formato binário que segue 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: `idPessoa`, `nomePessoa`, `idadePessoa`, `twitterPessoa`. 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 ("). Para a manipulação de *strings* com aspas duplas, pode-se usar a função scan_quote_string disponibilizada na página do projeto da disciplina.

Saída caso o programa seja executado com sucesso:

`Listar os arquivos binários arquivoPessoa.bin e arquivoIndexaPessoa.bin usando a função binarioNaTela.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
4 arquivoPessoa.bin arquivoIndexaPessoa.bin 2
29 "ADRIANA PEREIRA SILVA" NULO "DRI2345VA"
11 "MARIA DO NASCIMENTO" 15 "MADONA15"
```

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

[5] Permita a atualização de um ou mais campos de um ou mais registros do arquivo de dados **pessoa** que foram recuperados por meio da funcionalidade [3]. Quando necessário, deve ser realizado o tratamento de truncamento de dados. Nesses casos, devem ser considerados os *bytes* iniciais do dado sendo informado. Os *bytes* finais que não couberem no tamanho do campo são descartados. Campos com valores nulos, na entrada da funcionalidade, devem ser identificados com NULO. O lixo no registro atualizado, quando ele tiver tamanho menor do que o registro antes de ser atualizado, deve ser preenchido com o caractere '$'. Campos marcados como removidos não devem ser atualizados. Se o campo **idPessoa** for atualizado, então o arquivo de índice **indexaPessoa** tem que ser atualizado também. A funcionalidade [5] deve ser executada *n* vezes seguidas. Em situações nas quais não sejam encontrados registros, 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. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída dos arquivos binários. Lembre-se de manipular o campo “status” do registro de cabeçalho adequadamente. Primeiro ele deve começar com o valor ‘0’ (arquivo inconsistente) e, só ao final da execução do programa e ao final de todas as gravações, ele deve mudar para o valor ‘1’ (arquivo consistente).

Entrada do programa para a funcionalidade [5]:

`5 arquivoPessoa.bin arquivoIndexaPessoa.bin n`

$nomeCampo_{11}$ $valorCampo_{11}$ m_1 [$nomeCampo_{12}$ $valorCampo_{12}$ [...
$nomeCampo_{21}$ $valorCampo_{21}$ m_2 [$nomeCampo_{22}$ $valorCampo_{22}$ [...
...
$nomeCampo_{n1}$ $valorCampo_{n1}$ m_n [$nomeCampo_{n2}$ $valorCampo_{n2}$ [...

onde:

- `arquivoPessoa.bin` é o arquivo de dados **pessoa** gerado conforme as especificações descritas neste trabalho prático.

- `arquivoIndexaPessoa.bin` é o arquivo de índice **indexaPessoa** gerado conforme as especificações descritas neste trabalho prático.

- `n` é o número de atualizações a serem realizadas. Para cada atualização, deve ser informado o nome e o valor do campo que está sendo usado na busca
- `m` é o número de vezes que `nomeCampo` e `valorCampo` podem repetir na sintaxe do comando, desde que vários campos de um mesmo registro podem ser atualizados de uma única vez.

Cada uma das `n` atualizações deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre cada parâmetro de entrada. 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.

Saída caso o programa seja executado com sucesso:

`Listar os arquivos binários arquivoPessoa.bin e arquivoIndexaPessoa.bin usando a função binarioNaTela.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
5 arquivoPessoa.bin arquivoIndexaPessoa 1
idPessoa 25 2 nomePessoa "Samantha Santos" idadePessoa 15
```

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

Restrições

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

[1] Os arquivos de dados devem ser gravados 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] É necessário realizar o tratamento de truncamento de dados, conforme as instruções definidas.

[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. Para se fazer a busca, é possível caminhar no arquivo registro a registro, já que se sabe o tamanho do registro.

[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 nas páginas 96 a 107 do 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 por volta de 3 minutos de gravação (um máximo de 5 minutos é permitido, mas o ideal é o vídeo ter por volta de 3 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.

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. A entrega deve ser feita via [run.codes]:

- página: https://run.codes/Users/login
- código de matrícula: **V9DB**
- O vídeo gravado deve ser carregado no drive compartilhado SCC0607 Estrutura de Dados III > Trabalhos Práticos > Trabalho Prático T1. O vídeo deve ser nomeado como **Grupo X Video** (onde X representa o número do grupo).

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.

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 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.

Bom Trabalho !