Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Estrutura de Dados III

Docente
Prof. Anderson Canale Garcia
andersongarcia@usp.br

Monitores
Beatriz Aimée Teixeira Furtado Braga
beatrizatfb@usp.br ou telegram: @bia_aimee
Gustavo Lelli
gustavo.lelli@usp.br ou *telegram: @gustavo_lelli*
Lucas Piovani
lucaspiovani@usp.br ou telegram: @lucaspiovanii
Rafael Freitas Garcia
rafaelfreitasgarcia@usp.br ou *telegram: @rafaelfrgc*

Primeiro Trabalho Prático

Este trabalho tem como objetivo armazenar dados em um arquivo binário, bem como desenvolver funcionalidades para a manipulação desses dados.

O trabalho deve ser feito em duplas (ou seja, em 2 alunos). 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 Estrutura de Dados III é uma disciplina fundamental para a disciplina de Bases de Dados. A definição deste primeiro trabalho prático é feita considerando esse aspecto, ou seja, o projeto é especificado em termos de várias funcionalidades, e essas funcionalidades são 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 deste trabalho prático, os alunos poderão entrar em contato com alguns comandos da linguagem SQL e verificar como eles são implementados.

SQL é caracterizada por ser uma linguagem de definição de dados (DDL) e uma linguagem de manipulação de dados (DML). Comandos DDL são aqueles definidos utilizando, por exemplo, CREATE, DROP e ALTER. Comandos DML são aqueles definidos utilizando SELECT, INSERT, DELETE e UPDATE. Alguns desses comandos estarão presentes ao longo da especificação deste trabalho.

Os trabalhos práticos têm como objetivo armazenar, recuperar, manipular e visualizar dados relacionados aos dinossauros que aparecem no filme Jurassic Park e suas dietas. Esses dados são adaptados do dataset disponível em: https://www.kaggle.com/datasets/kjanjua/jurassic-park-the-exhaustive-dinosaur-dataset.

Descrição de páginas de disco

No trabalho será usado o conceito de páginas de disco. Cada página de disco tem o tamanho fixo de 1600 bytes. Como pode ser observado, o tamanho da página de disco adotado neste trabalho é lógico, sendo adaptado de acordo com os registros de dados do trabalho.

Descrição do Arquivo de Dados cadeiaAlimentar

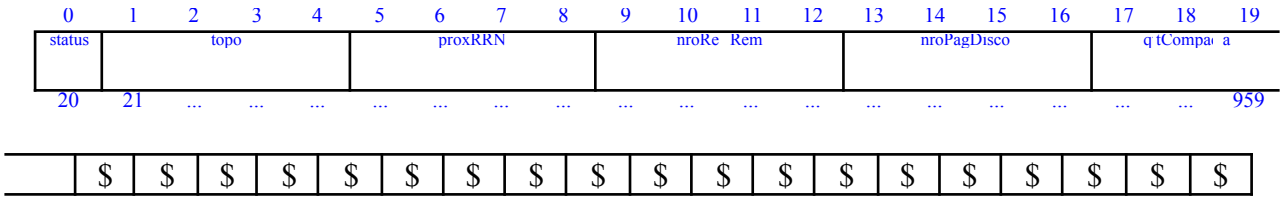
Um arquivo de dados possui um registro de cabeçalho e 0 ou mais registros de dados. A descrição do registro de cabeçalho é feita conforme a definição a seguir.

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 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.
- *topo:* armazena o RRN de um registro logicamente removido, ou -1 caso não haja registros logicamente removidos – tamanho: inteiro de 4 bytes.

- *proxRRN*: armazena o valor do próximo RRN disponível. Deve ser iniciado com o valor '0' e ser incrementado quando necessário – tamanho: inteiro de 4 bytes.
- *nroRegRem*: armazena o número de registros logicamente marcados como removidos. Deve ser iniciado com o valor '0' e deve ser atualizado sempre que necessário – tamanho: inteiro de 4 bytes.
- *nroPagDisco:* armazena o número de páginas de disco ocupadas pelo arquivo de dados, de acordo com a definição de página de disco feita neste trabalho. Deve ser iniciado com o valor '0' e deve ser atualizado sempre que necessário – tamanho: inteiro de 4 bytes.
- *qttCompacta:* indica a quantidade de vezes que o arquivo de dados foi compactado. Deve ser iniciado com o valor '0' e deve ser atualizado sempre que a funcionalidade de compactação for solicitada – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser o de uma página de disco, representado da seguinte forma:



Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Os campos são de tamanho fixo. Portanto, os valores que forem armazenados não devem ser finalizados por '\0'.
- O registro de cabeçalho deve ocupar uma página de disco. Portanto, descontando os campos de tamanho fixo definidos para o registro de cabeçalho, o espaço que sobra deve ser preenchido com o valor '$', o qual representa lixo.

A descrição dos registros de dados é feita conforme a definição a seguir.

Registros de Dados. Os registros de dados são de tamanho fixo, com campos de tamanho fixo e campos de tamanho variável. Para os campos de tamanho variável, deve ser usado o método delimitador entre campos. O delimitador é o caractere '#'.

Os campos de tamanho fixo são definidos da seguinte forma:

- *populacao*: quantidade de indivíduos da espécie que viviam em um determinado lugar – inteiro – tamanho: 4 bytes. Pode assumir valores nulos. Pode ter valores repetidos.
- *tamanho*: tamanho do indivíduo – float – tamanho: 4 bytes. Pode assumir valores nulos. Pode ter valores repetidos.
- *unidadeMedida*: unidade de medida da velocidade do individuo – tamanho: *string* de 1 byte. Pode assumir valores nulos. Pode ter valores repetidos.
- *velocidade*: velocidade do indivíduo – inteiro – tamanho: 4 bytes. Pode assumir valores nulos. Pode ter valores repetidos.

Os campos de tamanho variável são definidos da seguinte forma:

- *nome*: nome geral da espécie - string. Não pode assumir valores nulos, garantido pelo csv.. Não pode ter valores repetidos, garantido pelo csv.
- *especie*: nome da espécie – string. Não pode assumir valores nulos, garantido pelo csv. Pode ter valores repetidos.
- *habitat*: país em que a espécie vivia – string. Pode assumir valores nulos. Pode ter valores repetidos.
- *tipo*: categoria da espécie – string. Pode assumir valores nulos. Pode ter valores repetidos.
- *dieta*: tipo de alimentação da espécie – string. Não pode assumir valores nulos , garantido pelo csv. Pode ter valores repetidos.
- *alimento*: nome da espécie que serve de alimento para a principal – string. Pode assumir valores nulos. Pode ter valores repetidos.

Adicionalmente, os seguintes campos de tamanho fixo também compõem cada registro. Esses campos de controle são necessários para o gerenciamento de registros logicamente removidos.

- *removido*: indica se o registro está logicamente removido. Pode assumir os valores '1', para indicar que o registro está marcado como logicamente removido, ou '0', para indicar que o registro não está marcado como removido. –

tamanho: *string* de 1 byte.

- *encadeamento:* armazena o RRN do próximo registro logicamente removido – tamanho: inteiro de 4 bytes. Deve ser inicializado com o valor -1 quando necessário. Os dados são fornecidos juntamente com a especificação deste trabalho prático por meio de um arquivo .csv, sendo que sua especificação está disponível na página da disciplina. No arquivo .csv, o separador de campos é vírgula (,) e o primeiro registro especifica o que cada coluna representa (ou seja, contém a descrição do conteúdo das colunas).

Representação Gráfica dos Registros de Dados. O tamanho dos registros de dados deve ser de 160 bytes. Cada registro de dados deve ser representado da seguinte forma:

1 byte	4 bytes	4 bytes	4 bytes	1 byte	4 bytes		
removido	*encadeamento*	*populacao*	*tamanho*	*unidade Medida*	*velocidade*	*nome*	*especie*
0	1 ... 4	5 ... 8	9... 12	13	14 ... 17		...

habitat	*tipo*	*dieta*	*alimento*

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- As *strings* de tamanho variável são identificadas pelo seu tamanho e, portanto, não devem ser finalizadas com '\0'.
- Os valores nulos nos campos de tamanho fixo devem ser manipulados da seguinte forma. Os valores nulos devem ser representados pelo valor -1 quando forem inteiros ou devem ser totalmente preenchidos pelo lixo '$' quando forem do tipo *string*.
- Os valores nulos nos campos de tamanho variável devem ser manipulados da seguinte forma: deve ser armazenado apenas o delimitador '#'.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Sempre que

houver lixo, ele deve ser identificado pelo caractere '$'. Nenhum *byte* do registro deve permanecer vazio, ou seja, cada *byte* deve armazenar um valor válido ou '$'.

- Não existe a necessidade de truncamento dos dados. O arquivo .csv com os dados de entrada já garante essa característica.
- Os registros de dados não devem ser armazenados na mesma página de disco que o registro de cabeçalho. Adicionalmente, os registros de dados devem ser armazenados em várias páginas de disco, de acordo com a quantidade de registros gerados.

Programa

Descrição Geral. Implemente um programa em C por meio do qual o usuário possa obter dados de um arquivo de entrada e gerar arquivos binários com esses dados, bem como realizar operações de manipulação de dados nesses arquivos binários.

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 na definição das funcionalidades.

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, uma tabela possui um nome (que corresponde ao nome do arquivo) e várias colunas, as quais correspondem aos campos dos registros do arquivo de dados. A funcionalidade [1] representa um exemplo de implementação do comando CREATE TABLE.

[1] Permita a leitura de vários registros obtidos a partir de um arquivo de entrada no formato csv e a gravação desses registros em um arquivo de dados de saída. O arquivo de entrada no formato csv é fornecido juntamente com a especificação do projeto, enquanto o arquivo de dados de saída deve ser gerado de acordo com as especificações deste trabalho prático. O arquivo de dados de entrada não contém registros removidos. 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.

Entrada do programa para a funcionalidade [1]:

```
1 arquivoEntrada.csv arquivoSaida.bin
```

onde:

- `arquivoEntrada.csv` é um arquivo .csv que contém os valores dos campos dos registros a serem armazenados no arquivo binário.
- `arquivoSaida.bin` é o arquivo binário gerado conforme as especificações descritas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

```
Listar o arquivo 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
1 dinossauros.csv dinossauros.bin
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo dinossauros.bin
```

Na linguagem SQL, o comando SELECT é usado para listar os dados de uma tabela. Existem várias cláusulas que compõem o comando SELECT. O comando mais básico consiste em especificar as cláusulas SELECT e FROM, da seguinte forma:

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)

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

A funcionalidade [2] representa um exemplo de implementação do comando SELECT. Como todos os registros devem ser recuperados nessa funcionalidade, sua implementação consiste em percorrer sequencialmente o arquivo.

[2] Permita a recuperação dos dados de todos os registros armazenados em um arquivo de dados de entrada, 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. O arquivo de dados de entrada deve ser percorrido apropriadamente.

Entrada do programa para a funcionalidade [2]:

`2 arquivoEntrada.bin`

onde:

- `arquivoEntrada.bin` é o arquivo binário de entrada, o qual foi gerado conforme as especificações descritas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

Cada campo de cada registro deve ser mostrado em uma única linha contendo "descrição detalhada do campo: valor do campo". Caso o campo seja nulo, então não devem ser mostrados sua descrição detalhada e seu valor. Quanto à velocidade, deve ser colocado o valor da velocidade e o valor da medida de velocidade, o qual deve ser concatenado com a *string* "m/h", gerando km/h, hm/h ou cm/h. Deixe uma linha em branco depois que todos os campos de um registro forem exibidos. A ordem de exibição dos campos bem como os campos que devem ser exibidos podem ser vistos no **exemplo de execução**. No final, mostre o número de páginas de disco do arquivo de dados (páginas de disco referentes ao registro de cabeçalho e aos registros de dados).

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 (é mostrado um exemplo ilustrativo)**:**

```
./programaTrab
2 dinossauros.bin
Saída
Nome: tyrannosaurus
Especie: rex
Tipo: large theropod
Dieta: carnivorous
Lugar que habitava: USA
Tamanho: 12.0 m
Velocidade: 48 km/h
**** linha em branco ****
Numero de paginas de disco: 2

**** linha em branco ****
```

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. Como não existe índice definido sobre os campos dos registros, a implementação dessa funcionalidade consiste em percorrer sequencialmente o arquivo.

[3] 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, apenas um campo pode ser utilizado na busca. Por exemplo, é possível realizar a busca considerando somente o campo *nome* ou somente o campo *especie*. 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. A funcionalidade [3] 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 buscado, o programa deve continuar a executar as buscas até completar as *n* vezes seguidas. 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. O arquivo de dados de entrada deve ser percorrido apropriadamente. Quando encontrado um campo que é único, interrompe-se a busca específica.

Sintaxe do comando para a funcionalidade [3]:

```
3 arquivoEntrada.bin n
nomeCampo1 valorCampo1
nomeCampo2 valorCampo2
...
nomeCampon valorCampon
```

onde:

- `arquivoEntrada.bin` é o arquivo binário de entrada, o qual foi gerado conforme as especificações descritas neste trabalho prático.
- `n` é a quantidade de vezes que a busca deve ser realizada. Para cada busca, deve ser indicado o `nome do Campo` e o `valor do Campo` utilizados na busca. Cada um dos `n` (`nomeCampo valorCampo`) deve ser especificado em uma linha diferente. Deve ser deixado um espaço em branco entre `nomeCampo` e `valorCampo`. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

Para cada valor de n, exiba a saída conforme especificado na funcionalidade 2. No final, mostre o número de páginas de disco acessadas na busca n (páginas de disco referentes ao registro de cabeçalho e aos registros de dados).

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
3 dinossauros.bin 2 nome
"megalodon"
habitat "USA"
```

Saída

```
Busca 1
Registro Inexistente
**** linha em branco ****
Numero de paginas de disco: 55
**** linha em branco ****
Busca 2
Nome: apatosaurus
Especie: ajax
Tipo: sauropod
Dieta: herbivorous
Lugar que habitava: USA
Tamanho: 21.0 m
Velocidade: 14 km/h
**** linha em branco ****
Numero de paginas de disco: 55
**** linha em branco ****
```

[4] Permita a remoção lógica de registros em um arquivo de dados de entrada, baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve ser realizada usando o conceito de *lista de registros logicamente removidos*, e deve seguir estritamente a matéria apresentada em sala de aula. Como o arquivo possui registros de tamanho fixo, a lista deve ser implementada como uma pilha. Os registros a serem removidos devem ser aqueles que satisfaçam um critério de busca determinado pelo usuário, de acordo com a especificação da funcionalidade [3]. Note que qualquer campo pode ser utilizado como forma de remoção. Esta funcionalidade pode remover 0 registros (quando nenhum satisfaz ao critério de busca), 1 registro (quando apenas um satisfaz ao critério de busca), ou vários registros (quando vários satisfazem ao critério de busca). Ao se remover um registro, os valores dos *bytes* referentes aos campos já armazenados no registro devem ser preenchidos com o caractere '$', ou seja, com lixo, com exceção dos valores dos campos de controle. A funcionalidade [4] 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. 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 dados.

Entrada do programa para a funcionalidade [4]:

```
4 arquivoEntrada.bin n
nomeCampo1 valorCampo1
nomeCampo2 valorCampo2
...
nomeCampon valorCampon
```

onde:

- `arquivoEntrada.bin` é o arquivo de binário de entrada que foi gerado conforme as especificações descritas no primeiro trabalho prático. As remoções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `n` é a quantidade de vezes que a remoção deve ser realizada. Para cada remoção, deve ser indicado o `nome do Campo` e o `valor do Campo` utilizados na busca. Cada um dos `n` (`nomeCampo valorCampo`) deve ser especificado em uma linha diferente. Deve ser deixado um espaço em branco entre `nomeCampo` e `valorCampo`. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

```
Listar o arquivo 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
4 dinossauros.bin 1
especie "rex"
```

Saída

```
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo dinossauros.bin, o qual
foi atualizado frente às 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 [5] representa exemplo de implementação do comando INSERT INTO.

[5] Permita a inserção de novos registros em um arquivo de dados de entrada de um determinado tipo, baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve ser realizada usando o conceito de *lista de registros logicamente removidos*, e deve seguir estritamente a matéria apresentada em sala de aula. Como o arquivo possui registros de tamanho fixo, a lista deve ser implementada como uma pilha. O lixo que permanece no registro logicamente removido e que é reutilizado deve ser identificado pelo caractere '$'. Na entrada desta funcionalidade, os dados são referentes aos seguintes campos, na seguinte ordem: *nome*, *dieta*, *habitat*, *populacao*, *tipo*, *velocidade, medidaVelocidade, tamanho, especie, alimento*. Campos com valores nulos, na entrada da funcionalidade, devem ser identificados com NULO. 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. A funcionalidade [5] 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 do arquivo de dados binário.

Entrada do programa para a funcionalidade [5]:

`5 arquivoEntrada.bin n`

`idConecta1 nomePoPs1 nomePais1 siglaPais1 idPoPsConectado1 medidaVelocidade1 velocidade1`

`idConecta2 nomePoPs2 nomePais2 siglaPais2 idPoPsConectado2 medidaVelocidade2 velocidade2`

`...`

`idConectan nomePoPsn nomePaisn siglaPaisn idPoPsConectadon medidaVelocidaden velocidaden`

onde:

- `arquivoEntrada.bin` é o arquivo binário de entrada, o qual foi gerado conforme as especificações descritas neste trabalho prático. As inserções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `n` é o número de inserções a serem realizadas. 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 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 dinossauros.bin 2
“plateosaurus” “herbivorous” 58 "sauropod" 30 “h” 7.0 “engelhardti” “Equisetum arvense”
“shanag” "carnivorous" NULO NULO NULO NULO NULO NULO “ashile” NULO
```

usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo dinossauros.bin, o qual foi atualizado frente às inserções.

[6] Permita a compactação eficiente (desfragmentação) do arquivo de dados. A compactação elimina os registros removidos, deixando no arquivo de dados somente os registros marcados como não removidos. A compactação não deve eliminar o lixo, ou seja, não deve eliminar o caractere '$'. Como entrada dessa funcionalidade, tem-se um arquivo de dados binário que pode conter (ou não) registros logicamente removidos. Como saída, tem-se um arquivo de dados binário sem registros marcados como removidos. 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.

Entrada do programa para a funcionalidade [6]:

```
6 arquivoEntrada.bin
```

onde:

- `arquivoEntrada.bin` é o arquivo binário de entrada, o qual foi gerado conforme as especificações descritas neste trabalho prático. A compactação a ser realizada nessa funcionalidade deve ser feita nesse arquivo.

Saída caso o programa seja executado com sucesso:

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

Mensagem de saída caso algum erro seja encontrado:

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

Exemplo de execução:

```
./programaTrab1
6 dinossauros.bin
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo dinossauros.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. Outras bibliotecas podem ser utilizadas. 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.
- Uma especificação da participação de cada integrante no desenvolvimento do trabalho. Isso deve ser feito logo no início do arquivo que contém a função **main**. Deve ser indicado, para cada integrante, seu número USP, seu nome completo e sua porcentagem de participação. A porcentagem de participação deve variar entre 0% e 100%. Por exemplo, o integrante desenvolveu 100% do que era esperado de sua parte, ou então o integrante desenvolveu 80% do que era esperado para a sua parte.
- 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 de entrega.

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

- código de matrícula: K82G

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. Não deixem o vídeo em “Comentários”. Envie um arquivo txt.

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 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!