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
cdac@icmc.usp.br

Monitores
Gabriel Vicente Rodrigues
gabriel_vr@usp.br ou telegram:@ga_vr
Lucas de Medeiros Franca Romero
lucasromero@usp.br ou telegram: @lucasromero

Voluntário
João Paulo Clarindo
jpcsantos@usp.br

Primeiro Trabalho Prático

Este trabalho tem como objetivo armazenar dados em um arquivo binário de acordo com uma organização de campos e registros, bem como desenvolver funcionalidades para recuperar, inserir, remover e atualizar dados.

O trabalho deve ser feito por, no máximo, 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 será especificado em termos de várias funcionalidades, e essas funcionalidades serão relacionadas com as funcionalidades da linguagem SQL (*Structured Query Language*), que é a linguagem utilizada por sistemas gerenciadores de banco de dados (SGBDs) relacionais. As características e o detalhamento de SQL serão aprendidos na disciplina de Bases de Dados. Porém, por meio do desenvolvimento deste trabalho prático, os alunos poderão entrar em contato

desde a disciplina de Estrutura de Dados III com alguns comandos da linguagem SQL e verificar como eles são implementados.

Os trabalhos práticos têm como objetivo armazenar e recuperar dados relacionados às estações e linhas do metrô e da CPTM (Companhia Paulista de Trens Metropolitanos) da região metropolitana da cidade de São Paulo (SP). Um exemplo de uso desses dados é: "Peguei um ônibus de São Carlos para São Paulo. O ônibus chega na Rodoviária do Tietê. Eu preciso pegar uma linha de metrô para ir para o Aeroporto de Guarulhos." Os dados manipulados indicam o trajeto que deve ser feito.

Em detalhes, os dados se referem às estações, às linhas, às distâncias entre as estações e aos trajetos. Na disciplina de Bases de Dados, é ensinado que esses dados devem ser armazenados em diferentes arquivos, de acordo com a normalização realizada. Entretanto, para simplificação dos trabalhos da disciplina de Estrutura de Dados III, todos os dados serão armazenados em apenas um arquivo. Isso significa que existe redundância dos dados, ou seja, o mesmo valor de dados é armazenado mais do que uma vez.

Descrição do arquivo de dados estacao

O arquivo de dados **estacao** 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.
- *topoLista:* armazena o *byte offset* de um registro logicamente removido, ou -1 caso não haja registros logicamente removidos – tamanho: inteiro de 8 bytes.

- *nroEstacoes*: indica a quantidade de estações diferentes que estão armazenadas no arquivo de dados. Note que, se duas ou mais estações têm o mesmo nome, elas são consideradas a mesma estação – tamanho: *inteiro* de 4 bytes.
- *nroParesEstacao*: indica a quantidade de pares (codEstacao, codProxEstacao) diferentes que estão armazenados no arquivo de dados – tamanho: *inteiro* de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 17 bytes, representado da seguinte forma:

1 byte	8 bytes	4 bytes	4 bytes
status	*topoLista*	*nroEstacoes*	*nroParesEstacao*
0	1…8	9…12	13…16

Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Os campos são de tamanho fixo e, na maioria, do tipo inteiro. Portanto, os valores que forem armazenados não devem ser finalizados por '\0'.
- Neste projeto, o conceito de página de disco não está sendo considerado.

Registros de Dados. Os registros de dados são de tamanho variável, com campos de tamanho fixo e campos de tamanho variável. Para os campos de tamanho variável, deve ser usado o método delimitador entre campos, sendo esse delimitador o pipe (|). Para os registros de tamanho variável, deve ser usado o método indicador de tamanho.

Os campos de tamanho fixo são definidos da seguinte forma:

- *codEstacao*: código da estação – inteiro – tamanho: 4 bytes.
- *codLinha*: código da linha – inteiro – tamanho: 4 bytes.
- *codProxEstacao*: código da próxima estação – inteiro – tamanho: 4 bytes.
- *distProxEstacao*: distância para a próxima estação – inteiro – tamanho: 4 bytes.
- *codLinhaIntegra*: código da linha que faz a integração entre as linhas – inteiro – tamanho: 4 bytes.

- *codEstIntegra*: código da estação que faz a integração entre as linhas – inteiro – tamanho: 4 bytes.

Os campos de tamanho variável são definidos da seguinte forma:

- *nomeEstacao*: nome da estação
- *nomeLinha*: nome da linha

Adicionalmente, os seguintes campos de tamanho fixo também compõem cada registro. Esses campos são necessários para o gerenciamento de registros logicamente removidos e para oferecer suporte para o método indicador de tamanho.

- *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.
- *tamanhoRegistro:* número de bytes do registro – inteiro – tamanho: 4 bytes.
- *proxLista:* armazena os *byte offsets* dos registros logicamente removidos – tamanho: inteiro de 8 bytes

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 (,). Note que, no arquivo .csv, os dados são referentes aos seguintes campos, na seguinte ordem: codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra.

Representação Gráfica dos Registros de Dados. Cada registro de dados deve ser representado da seguinte forma:

1 byte	4 bytes	8 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	variável	variável
removido	*tamanho Registro*	*prox Lista*	*cod Estacao*	*cod Linha*	*codProx Estacao*	*distProx Estacao*	*codLinha Integra*	*codEst Integra*	*nome Estacao*	*nome Linha*
0	1...4	5...12	13...16	17...20	21...24	25...28	29...32	33...36	37 ...	...

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- As *strings* de tamanho variável devem ser delimitadas pelo caractere pipe e, portanto, não devem ser finalizadas com '\0'.
- Os campos *codEstacao* e *nomeEstacao* não aceitam valores nulos. Os demais campos aceitam valores nulos. O arquivo .csv com os dados de entrada já garante essa característica.
    - Para os campos de tamanho fixo, os valores nulos devem ser representados pelo valor -1.
    - Para os campos de tamanho variável, armazenar um valor nulo significa armazenar apenas o delimitador pipe.
- 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.
- Neste projeto, o conceito de página de disco não está sendo considerado.

Programa

Descrição Geral. Implemente um programa em C por meio do qual o usuário possa obter dados de um arquivo de entrada e gerar um arquivo binário com esses dados, bem como realizar operações de busca, inserção, remoção e atualização dos dados contidos nesse arquivo binário.

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, 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 do arquivo de entrada **estacao.csv** e a gravação desses registros em um arquivo de dados de saída. O arquivo de entrada **estacao.csv** é fornecido juntamente com a especificação do projeto, enquanto que o arquivo de dados de saída deve ser gerado como parte deste trabalho prático. Note que, no arquivo estacao.csv, os dados são referentes aos seguintes campos, na seguinte ordem: codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra. 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 estacao.csv estacao.bin`

onde:

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

Saída caso o programa seja executado com sucesso:

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

[2] Permita a recuperação dos dados de todos os registros armazenados no arquivo de dados **estacao.bin**, 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 estacao.bin`

onde:

- `estacao.bin` é o arquivo binário gerado conforme as especificações descritas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

Cada registro deve ser mostrado em uma única linha e os seus campos devem ser mostrados de forma sequencial separado por um espaço em branco. Campos de tamanho fixo que tiverem o valor nulo devem ser exibidos da seguinte forma: ao invés de exibir o valor -1, escreva NULO. Campos de tamanho variável que tiverem o valor nulo devem ser exibidos da seguinte forma: NULO. A ordem de exibição dos campos dos registros deve ser codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra, conforme ilustrado no **exemplo de execução.**

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 estacao.bin
1 Tucuruvi 1 Azul 2 992 NULO NULO
2 Parada Inglesa 1 Azul 3 1057 4 55
```

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.

[3] Permita a recuperação dos dados de todos os registros do arquivo **estacao.bin** que 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 *codEstacao* ou considerando os campos *nomeEstacao* e *nomeLinha*. 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 [3]:

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

onde:

- `estacao.bin` é o arquivo binário gerado conforme as especificações descritas neste trabalho prático.

- `n` é a quantidade de vezes que `nome do Campo` e `valor do Campo` podem repetir na sintaxe do comando. 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:

`Cada registro deve ser mostrado em uma única linha e os seus campos devem ser mostrados de forma sequencial separado por um espaço em branco. Campos de tamanho fixo que tiverem o valor nulo devem ser exibidos da seguinte forma: ao invés de exibir o valor -1, escreva NULO. Campos de tamanho variável que tiverem o valor nulo devem ser exibidos da seguinte forma: NULO. A ordem de exibição dos campos dos registros deve ser codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra, conforme ilustrado 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
3 estacao.bin 1
nomeEstacao "Luz"
9 Luz 1 Azul 10 NULO 4 55
55 Luz 4 Amarela 56 1257 1 9
111 Luz 7 Rubi 112 NULO NULO NULO
```

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 [4] representa um exemplo de implementação do comando DELETE.

[4] Permita a remoção lógica de registros, baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve ser realizada usando o conceito de *lista encadeada de registros logicamente removidos*, e deve seguir estritamente a matéria apresentada em sala de aula. Os registros a serem removidos devem ser aqueles que satisfaçam um critério de busca determinado pelo usuário. Por exemplo, o usuário pode solicitar a remoção de um registro que possui um determinado *codEstacao* ou o usuário pode solicitar a remoção de todos os registros que possuam *nomeEstacao* igual a *Luz*. Note que qualquer campo pode ser utilizado como forma de remoção. Ao se remover um registro, os valores dos *bytes* referentes aos campos já armazenados no registro devem permanecer os mesmos, com exceção dos valores dos campos relacionados ao tratamento da lista encadeada. 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.

Entrada do programa para a funcionalidade [4]:

```
4 estacao.bin n
x1 nomeCampo1 valorCampo1 … nomeCampox1 valorCampox1
x2 nomeCampo1 valorCampo1 … nomeCampox2 valorCampox2
...
xn nomeCampo1 valorCampo1 … nomeCampox2 valorCampoxn
```

onde:

- `estacao.bin` é um arquivo binário gerado conforme as especificações descritas neste trabalho prático. As remoções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `n` é o número de remoções a serem realizadas. Para cada remoção, deve ser informado em uma linha diferente o número de campos `x` usados como critério de busca e, para cada campo usado como critério de busca, o `nome do campo` e o seu respectivo `valor do campo`. Deve ser deixado um espaço em branco entre o `nome do campo` e o `valor do campo`. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

```
Listar o arquivo binário estacao.bin.
```

Mensagem de saída caso algum erro seja encontrado:

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

Exemplo de execução:

```
./programaTrab
4 estacao.bin 2
1 nomeEstacao "Luz"
2 codLinha 7 codProxEst 2
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo estacao.bin, o qual foi atualizado 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.

INSERT INTO tabela

VALUES valores a serem inseridos

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

[5] Permita a inserção de novos registros no arquivo de entrada **estacao.bin**, baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve ser realizada usando o conceito de *lista encadeada de registros logicamente removidos*, e deve seguir estritamente a matéria apresentada em sala de aula. 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: codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra. 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 binário.

Entrada do programa para a funcionalidade [5]:

`5 estacao.bin n`

`codEstacao`$_1$ `nomeEstacao`$_1$ `codLinha`$_1$ `nomeLinha`$_1$ `codProxEstacao`$_1$ `distProxEstacao`$_1$ `codLinhaIntegra`$_1$ `codEstacaoIntegra`$_1$

`codEstacao`$_2$ `nomeEstacao`$_2$ `codLinha`$_2$ `nomeLinha`$_2$ `codProxEstacao`$_2$ `distProxEstacao`$_2$ `codLinhaIntegra`$_2$ `codEstacaoIntegra`$_2$

`...`

`codEstacao`$_n$ `nomeEstacao`$_n$ `codLinha`$_n$ `nomeLinha`$_n$ `codProxEstacao`$_n$ `distProxEstacao`$_n$ `codLinhaIntegra`$_n$ `codEstacaoIntegra`$_n$

onde:

- `estacao.bin` é um arquivo binário de entrada que segue as mesmas especificações definidas nesse trabalho prático. As inserções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `n` é o número de inserções a serem realizadas. Para cada inserção, deve ser informado os valores a serem inseridos no arquivo, considerando os seguintes campos, na seguinte ordem: codEstacao, nomeEstacao, codLinha, nomeLinha, codProxEstacao, distProxEstacao, codLinhaIntegra, codEstIntegra. Valores nulos devem ser identificados, na entrada da funcionalidade, por NULO. Cada uma das `n` inserções deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre os valores dos campos. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

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

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
5 estacao.bin 2
500 "Teste" 10 "Branca" NULO NULO NULO
501 "Nova Estacao" 10 "Branca" NULO NULO NULO NULO
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo estacao.bin, o qual foi atualizado 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 [6] representa um exemplo de implementação do comando UPDATE.

[6] Permita a atualização de registros. Os registros a serem atualizados devem ser aqueles que satisfaçam um critério de busca determinado pelo usuário. Por exemplo, o usuário pode solicitar a atualização de um registro que possui um determinado *codEstacao* ou o usuário pode solicitar a atualização de todos os registros que possuam *nomeEstacao* igual a *Luz*. Note que qualquer campo pode ser utilizado como forma de atualização. Adicionalmente, o campo utilizado como busca não precisa ser, necessariamente, o campo a ser atualizado. Por exemplo, pode-se buscar pelo campo *codEstacao*, e pode-se atualizar o campo *nomeEstacao*. Quando o tamanho do registro atualizado for maior do que o tamanho do registro atual, então o registro atual deve ser logicamente removido e o registro atualizado deve ser inserido como um novo registro. A implementação dessa funcionalidade deve ser realizada usando o conceito de *lista encadeada de registros logicamente removidos*, e deve seguir estritamente a matéria apresentada em sala de aula. 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. 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 estacao.bin n`

`x1` $\texttt{nomeCampoBusca}_1$ $\texttt{valorCampoBusca}_1$ ... $\texttt{nomeCampoBuscax}_{x1}$ $\texttt{valorCampoBusca}_{x1}$
`y1` $\texttt{nomeCampoAtualiza}_1$ $\texttt{valorCampoAtualiza}_1$ ... $\texttt{nomeCampoAtualiza}_{y1}$ $\texttt{valorCampoAtualiza}_{y1}$
`x2` $\texttt{nomeCampoBusca}_1$ $\texttt{valorCampoBusca}_1$ ... $\texttt{nomeCampoBusca}_{x2}$ $\texttt{valorCampoBusca}_{x2}$
`y2` $\texttt{nomeCampoAtualiza}_1$ $\texttt{valorCampoAtualiza}_1$ ... $\texttt{nomeCampoAtualiza}_{y2}$ $\texttt{valorCampoAtualiza}_{y2}$
...
`xn` $\texttt{nomeCampoBusca}_1$ $\texttt{valorCampoBusca}_1$ ... $\texttt{nomeCampoBusca}_{xn}$ $\texttt{valorCampoBusca}_{xn}$
`yn` $\texttt{nomeCampoAtualiza}_1$ $\texttt{valorCampoAtualiza}_1$ ... $\texttt{nomeCampoAtualiza}_{yn}$ $\texttt{valorCampoAtualiza}_{yn}$

onde:

- `estacao.bin` é um arquivo binário de entrada que segue as especificações definidas neste trabalho prático. As atualizações a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `n` é o número de atualizações a serem realizadas. Para cada uma das `n` atualizações, devem ser informados em uma nova linha: (i) o número `x` de campos que são usados para buscar o registro, cada nome de campo e o seu respectivo valor; (ii) o número `y` de campos que serão atualizados, cada nome de campo e o respectivo valor. Valores nulos devem ser identificados, na entrada da funcionalidade, por NULO. O nome do campo de busca e o nome do campo a ser atualizado podem ser iguais ou diferentes. Deve ser deixado um espaço em branco entre cada um dos parâmetros de entrada. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

`Listar o arquivo binário arquivo.bin.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
6 estacao.bin 2
2 codEstacao 1 nomeEstacao "Tucuruvi"
3 codProxEstacao 15 codLinhaIntegrada 4 codEstacaoIntegrada 55
1 codEstacao 15
1 nomeEstacao "Exemplo"
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo estacao.bin, o qual foi atualizado com as atualizações.
```

Restrições

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

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

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

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

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

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

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

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

[8] Todo código fonte deve ser documentado. A **documentação interna** inclui, dentre outros, a documentação de procedimentos, de funções, de variáveis, de partes do código fonte que realizam tarefas específicas. Ou seja, o código fonte deve ser

documentado tanto em nível de rotinas quanto em nível de variáveis e blocos funcionais.

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

Fundamentação Teórica

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

Material para Entregar

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

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

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

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

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

Instruções de entrega.

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

- página: https://run.codes/Users/login
- código de matrícula: **QLAT**

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

Critério de Correção

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

- Corretude da execução do programa.
- Atendimento às especificações do registro de cabeçalho e dos registros de dados.
- Atendimento às especificações da sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade.

- Qualidade da documentação entregue. A documentação interna terá um peso considerável no trabalho.
- Vídeo. Integrantes que não participarem da apresentação receberão nota 0 no trabalho correspondente.

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

Restrições adicionais sobre o critério de correção.

- A não execução de um programa devido a erros de compilação implica que a nota final da parte do trabalho será igual a zero (0).
- O não atendimento às especificações do registro de cabeçalho e dos registros de dados implica que haverá uma diminuição expressiva na nota do trabalho.
- O não atendimento às especificações de sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade implica que haverá uma diminuição expressiva na nota do trabalho.
- A ausência da documentação implica que haverá uma diminuição expressiva na nota do trabalho.
- A realização do trabalho prático com alunos de turmas diferentes implica que haverá uma diminuição expressiva na nota do trabalho.
- A inserção de palavras ofensivas nos arquivos e em qualquer outro material entregue implica que a nota final da parte do trabalho será igual a zero (0).
- Em caso de plágio, as notas dos trabalhos envolvidos serão zero (0).

Data de Entrega do Trabalho

Na data especificada na página da disciplina.

Bom Trabalho!