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
Caio Capocasali da Silva
caio.capocasali@usp.br *ou telegram: @CaioCapocasali*
Renan Banci Catarin
renanbcatarin@usp.br *ou telegram: @Reckat*
Renan Trofino Silva
renan.trofino@usp.br *ou telegram: @renan823*

Primeiro Trabalho Prático

Este trabalho tem como objetivos armazenar dados em arquivos binários de acordo com uma organização de campos e registros, bem como realizar operações de recuperação, inserção, remoção, atualização e junção.

O trabalho deve ser feito por 2 alunos, que são os mesmos alunos do trabalho introdutório. Qualquer mudança deve ser autorizada pela docente. A solução deve ser proposta exclusivamente pelos alunos com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.

Fundamentos da disciplina de Bases de Dados

Conforme descrito no trabalho introdutório, os trabalhos práticos têm como objetivo armazenar e recuperar dados relacionados às *pessoas que seguem pessoas em uma rede social*. 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 trabalho, visa-se implementar aspectos de: (i) inserção, remoção e

atualização no arquivo de dados **pessoa**; (ii) criação e busca do arquivo de dados **segue**; (iii) junção entre os arquivos de dados **pessoa** e **segue**.

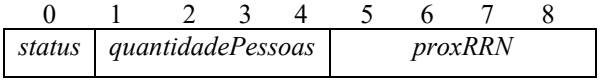
Descrição do Arquivo Segue

O arquivo de dados **segue** 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 que seguem outras pessoas (ou o número de registros) presentes no arquivo – tamanho: inteiro de 4 bytes. Deve ser iniciado com o valor '0' e deve ser incrementado e decrementado quando necessário – tamanho: inteiro de 4 bytes.
- *proxRRN*: armazena o valor do próximo *byte offset* disponível. Deve ser iniciado com o valor 0 e deve ser alterado sempre que necessário – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 9 bytes, representado da seguinte forma:



Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Neste projeto, o conceito de página de disco não está sendo considerado.

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.
- *idPessoaQueSegue*: código que identifica univocamente cada pessoa que segue outra pessoa – inteiro – tamanho: 4 bytes.
- *idPessoaQueESeguida:* código que identifica univocamente cada pessoa que é seguida por outra pessoa – inteiro – tamanho: 4 bytes.
- *dataInicioQueSegue*: data de início que a pessoa *idPessoaQueSegue* começou a seguir a pessoa *idPessoaQueESeguida* – *string* – tamanho: 10 bytes, no formato DD/MM/AAAA.
- *dataFimQueSegue*: data na qual a pessoa identificada por *idPessoaQueSegue* parou de seguir a pessoa *idPessoaQueESeguida* – *string* - tamanho: 10 bytes, no formato DD/MM/AAAA.
- *grauAmizade*: indica o grau de amizade que a pessoa que segue tem pela pessoa que é seguida. Pode assumir os valores '0' (segue porque é uma celebridade), '1' (segue porque é amiga de minha amiga), '2' (segue porque é minha amiga) – tamanho: *string* de 1 byte.

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). Adicionalmente, campos nulos são representados por espaço em branco.

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

0	1 2 3 4	5 6 7 8	9 … … 18	19 … … 28	29
removido	*idPessoaQueSegue*	*idPessoaQueESeguida*	*dataInicioQueSegue*	*dataFimQueSegue*	*grauAmizade*

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- 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 '$' quando forem do tipo *string*.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Sempre que houver lixo, ele deve ser identificado pelo caractere '$'. 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 arquivos binários com esses dados, bem como realizar operações de busca, inserção, remoção, atualização e junção.

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

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

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

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

[5] Permita a remoção lógica dos registros do arquivo de dados **pessoa**, baseado na *abordagem estática* de reaproveitamento de espaços de registros logicamente removidos. Os registros a serem removidos devem ser aqueles que satisfaçam um critério de busca determinado pelo usuário, sendo que a busca deve ser realizada conforme a especificação da funcionalidade [4]. Ao se remover um registro, os valores dos *bytes* referentes aos campos já armazenados no registro devem permanecer os mesmos. 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. 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 as saídas dos arquivos binários de dados e de índice.

Sintaxe do comando para a funcionalidade [5]:

`5 arquivoEntrada.bin arquivoIndicePrimario.bin n`

`1 nomeCampo`$_1$`=valorCampo`$_1$

`2 nomeCampo`$_2$`=valorCampo`$_2$

`...`

`n nomeCampo`$_n$`=valorCampo`$_n$

onde:

- `arquivoEntrada.bin` é o arquivo de dados **pessoa** no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo Pessoa** definidas no trabalho prático introdutório.

- `arquivoIndicePrimario.bin` é um arquivo de índice primário no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo de Índice Primário** definidas no trabalho prático introdutório.

- `n` é a quantidade de vezes que a remoção deve acontecer.

- `nomeCampo`= `valorCampo` referem-se ao nome e ao valor do campo que estão sendo procurados no arquivo de dados **pessoa** para remoção. Deve ser colocado um símbolo de igual (=) entre `nomeCampo` e `valorCampo`, sem espaço em branco. Campos com valores nulos devem ser identificados com NULO. Adicionalmente, os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

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 processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
5 pessoa.bin indexaPessoa.bin 2
1 idPessoa=25
2 idadePessoa=NULO
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos pessoa.bin e indexaPessoa.bin, os quais foram atualizados 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 [6] representa exemplo de implementação do comando INSERT INTO.

[6] Permita a inserção de novos registros no arquivo de dados **pessoa**, baseado na *abordagem estática* de reaproveitamento de espaços de registros logicamente removidos. Na entrada desta funcionalidade, os dados são referentes aos seguintes campos, na seguinte ordem: idPessoa, nomePessoa, idadePessoa, nomeUsuario. 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. Não será solicitada a inserção de valores nulos ou repetidos para os campos *idPessoa* e *nomeUsuario*. A funcionalidade [6] 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 as saídas dos arquivos binários de dados e de índice.

Sintaxe do comando para a funcionalidade [6]:

`6 arquivoEntrada.bin arquivoIndicePrimario.bin n`
`1 valor_idPessoa₁, valor_nomePessoa₁, valor_idade₁, valor_nomeUsuario₁`
`2 valor_idPessoa₂, valor_nomePessoa₂, valor_idade₂, valor_nomeUsuario₂`
`...`
`n valor_idPessoaₙ, valor_nomePessoaₙ, valor_idadeₙ, valor_nomeUsuarioₙ`

onde:

- `arquivoEntrada.bin` é o arquivo de dados **pessoa** no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo Pessoa** definidas no trabalho prático introdutório.
- `arquivoIndicePrimario.bin` é um arquivo de índice primário no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo de Índice Primário** definidas no trabalho prático introdutório.
- `n` é a quantidade de vezes que a inserção deve acontecer.

- `valor_idPessoa, valor_nomePessoa, valor_idade, valor_nomeUsuario` são os dados a serem inseridos nos seguintes campos, na seguinte ordem: idPessoa, nomePessoa, idadePessoa, nomeUsuario. Campos com valores nulos devem ser identificados com NULO. Adicionalmente, os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

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 processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
6 pessoa.bin indexaPessoa.bin 2
1 25, "SAMANTHA PEREIRA SANTOS", 13, "SAMANTHAPS"
2 45, "VITORIA PRADO CAMPOS", NULO, "VIVICA"
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos pessoa.bin e indexaPessoa.bin, os quais foram atualizados frente às inserções.`

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

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

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

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

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

[7] Permita a atualização dos registros do arquivo de dados **pessoa**, baseado na *abordagem estática* de reaproveitamento de espaços de registros logicamente removidos. Quando o tamanho do registro atualizado for maior do que o tamanho do registro atual, o registro atual deve ser logicamente removido e o registro atualizado deve ser inserido como um novo registro. 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 permanecer no registro atualizado deve ser identificado pelo caractere '$'. Os registros a serem atualizados devem ser aqueles que satisfaçam um critério de busca determinado pelo usuário, sendo que a busca deve ser realizada conforme a especificação da funcionalidade [4]. Note que o campo utilizado como busca não precisa ser, necessariamente, o campo a ser atualizado. Por exemplo, pode-se buscar pelo campo *idPessoa*, e pode-se atualizar o campo *idadePessoa*. Campos a serem atualizados com valores nulos devem ser identificados, na entrada da funcionalidade, 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. Não será solicitada a atualização dos campos *idPessoa* e *nomeUsuario* de com valores nulos ou repetidos. A funcionalidade [7] 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 as saídas dos arquivos binários de dados e de índice.

Entrada do programa para a funcionalidade [7]:

`7 arquivoEntrada.bin arquivoIndicePrimario.bin n`

`1` $\texttt{nomeCampoBusca}_1\texttt{=valorCampoBusca}_1$ $\texttt{nomeCampoAtualiza}_1\texttt{=valorCampoAtualiza}_1$

`2` $\texttt{nomeCampoBusca}_2\texttt{=valorCampoBusca}_2$ $\texttt{nomeCampoAtualiza}_2\texttt{=valorCampoAtualiza}_2$

`...`

`n` $\texttt{nomeCampoBusca}_n\texttt{=valorCampoBusca}_n$ $\texttt{nomeCampoAtualiza}_n\texttt{=valorCampoAtualiza}_n$

onde:

- `arquivoEntrada.bin` é o arquivo de dados **pessoa** no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo Pessoa** definidas no trabalho prático introdutório.
- `arquivoIndicePrimario.bin` é um arquivo de índice primário no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo de Índice Primário** definidas no trabalho prático introdutório.
- `n` é a quantidade de vezes que a atualização deve acontecer.
- `nomeCampo=valorCampo` referem-se ao nome e ao valor do campo que estão sendo procurados no arquivo de dados **pessoa**. Deve ser colocado um símbolo de igual (=) entre `nomeCampo` e `valorCampo`, sem espaço em branco. Campos com valores nulos devem ser identificados com NULO. Adicionalmente, os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").
- `nomeCampoAtualiza=valorCampoAtualiza` referem-se ao nome e ao valor do campo que vão ser atualizados no arquivo de dados **pessoa**. Deve ser colocado um símbolo de igual (=) entre `nomeCampoAtualiza` e `valorCampoAtualiza`, sem espaço em branco. Campos com valores nulos devem ser identificados com NULO. Adicionalmente, os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

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 processamento do arquivo.
```

Exemplo de execução:

```
./programaTrab
7 pessoa.bin indexaPessoa.bin 2
1 idPessoa=25 nomePessoa="MARIANA PEREIRA"
2 idadePessoa=13 idadePessoa=20
```

usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos pessoa.bin e indexaPessoa.bin, os quais foram atualizados frente às atualizações.

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 [8] representa um exemplo de implementação do comando CREATE TABLE.

[8] Permita a leitura de vários registros obtidos a partir de um arquivo de entrada no formato csv e a gravação desses registros no arquivo de dados de saída **segue**. O arquivo de entrada no formato csv é fornecido juntamente com a especificação do projeto, enquanto o arquivo de dados **segue** deve ser gerado de acordo com as especificações deste trabalho prático. Nesta funcionalidade, ocorrem várias inserções no arquivo **segue**. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar os dados do arquivo **segue**.

Entrada do programa para a funcionalidade [8]:

```
8 arquivoEntrada.csv arquivoSaida.bin
```

onde:

- `arquivoEntrada.csv` é um arquivo .csv que contém os valores dos campos dos registros a serem armazenados no arquivo de dados **segue**.
- `arquivoSaida.bin` é o arquivo de dados **segue** no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo de Dados Segue** definidas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

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

Na linguagem SQL, a cláusula ORDER BY é usada para ordenar os resultados de uma consulta em ordem crescente (padrão) ou decrescente, com base em uma ou mais colunas. Ela é especificada juntamente com o comando SELECT.

```
SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela (ou seja, arquivo)
WHERE ...  // opcional
ORDER BY lista de colunas (ou seja, campos a serem ordenados na resposta)
```

A funcionalidade [9] representa um exemplo de implementação da cláusula ORDER BY.

[9] Ordene o arquivo de dados **segue** de acordo com o campo de ordenação *idPessoaQueSegue*. Quando houver empate para os valores de *idPessoaQueSegue*, deve ser usado como critério de desempate o campo *idPessoaQueESeguida* e, caso ainda haja empate, deve ser usado como critério de desempate o campo *dataInicioQueSegue* e, caso ainda haja empate, deve ser usado como critério de desempate o campo *dataFimQueSegue*. Considere que a ordenação deve ser feita de forma crescente. Considere também que, para os campos que podem assumir valores nulos, o valor não nulo deve aparecer antes do que o valor nulo. A ordenação deve ser implementada considerando que o arquivo de dados **segue** cabe totalmente em memória primária (RAM). Portanto, o arquivo deve ser: (i) lido inteiramente do disco para a RAM; (ii) ordenado de forma crescente de acordo com a chave de ordenação usando-se qualquer algoritmo de ordenação disponível na biblioteca da linguagem C; e (iii) escrito inteiramente para disco novamente, gerando um novo arquivo de dados **segueOrdenado**, o qual encontra-se ordenado com base no campo de ordenação. O arquivo de dados **segue** original não deve ser removido.

Entrada do programa para a funcionalidade [9]:

`9 arquivoDesordenado.bin arquivoOrdenado.bin`

onde:

- `arquivoDesordenado.bin` é o arquivo de dados **segue** no formato binário conforme as especificações da **Descrição do Arquivo de Dados Segue** definidas neste trabalho prático, sem considerar ordenação.
- `arquivoOrdenado.bin` é o arquivo de dados **segue** no formato binário conforme as especificações da **Descrição do Arquivo de Dados Segue** definidas neste trabalho prático, considerando a ordenação.

Saída caso o programa seja executado com sucesso:

`Listar o arquivo de dados 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
9 segue.bin segueOrdenado.bin
```

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

A operação de junção "junta" dados de registros de dois arquivos usando com base em um campo de igualdade (ou seja, uma condição de junção). Na linguagem SQL, a junção pode ser especificada da seguinte forma:

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela1, tabela2 (ou seja, dois arquivos)
WHERE tabela1.campoIgualdade = tabela2.campoIgualdade
(condição de junção)

Outra forma de representar a junção na linguagem SQL é:

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela1 JOIN tabela2 (ou seja, dois arquivos)
ON tabela1.campoIgualdade = tabela2.campoIgualdade
(condição de junção)

A funcionalidade [10] representa um exemplo de implementação da operação de junção.

[10] Realize a junção do arquivo de dados **pessoa** com o arquivo de dados **segueOrdenado**, considerando como condição de junção o campo *idPessoa* do arquivo de dados **pessoa** e o campo *idPessoaQueSegue* do arquivo de dados **segueOrdenado**.

Existem várias formas de se implementar a junção em bancos de dados. Neste trabalho prático, ela deve ser implementada da seguinte forma. Primeiramente, utilize a funcionalidade [4] para buscar os dados dos registros do arquivo **pessoa** que satisfaçam um critério de busca determinado pelo usuário. Para cada registro recuperado, utilize o campo *idPessoa* para procurar no arquivo **segueOrdenado** os registros que satisfaçam à condição de junção de *idPessoa* = *idPessoaQueSegue*. O arquivo **segueOrdenado** encontra-se ordenado pelo campo *idPessoaQueSegue* e, portanto, deve ser utilizada busca binária para recuperar os dados solicitados.

Para cada registro recuperado do arquivo de dados **pessoa**, as seguintes situações podem ocorrer:

- Não existe igualdade entre *idPessoa* e *idPessoaQueSegue*. Nesse caso, nenhum registro é retornado como resultado da junção.
- Existe igualdade entre *idPessoa* e *idPessoaQueSegue* de forma que existe apenas um registro do arquivo **segueOrdenado**. Nesse caso, somente um registro é retornado como resultado da junção.
- Existe igualdade entre *idPessoa* e *idPessoaQueSegue*, de forma que existem vários (ou seja, *n*) registros do arquivo **segueOrdenado**. Neste caso, são retornados *n* registros como resultado da junção.

Sintaxe do comando para a funcionalidade [10]:

`10 arquivoEntrada.bin arquivoIndicePrimario.bin arquivoOrdenado.bin n`
`1 nomeCampo`$_1$`=valorCampo`$_1$
`2 nomeCampo`$_2$`=valorCampo`$_2$
`...`
`n nomeCampo`$_n$`=valorCampo`$_n$

onde:

- `arquivoEntrada.bin` é o arquivo de dados **pessoa** no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo Pessoa** definidas no trabalho prático introdutório.

- `arquivoIndicePrimario.bin` é um arquivo de índice primário no formato binário, o qual é gerado de acordo com as especificações da **Descrição do Arquivo de Índice Primário** definidas no trabalho prático introdutório.

- `arquivoOrdenado.bin` é o arquivo de dados **segue** no formato binário conforme as especificações da **Descrição do Arquivo de Dados Segue** definidas neste trabalho prático, considerando a ordenação gerada pela funcionalidade [9].

- `n` é a quantidade de vezes que a busca deve acontecer.

- `nomeCampo=valorCampo` referem-se ao nome e ao valor do campo que estão sendo procurados no arquivo de dados **pessoa**. Deve ser colocado um símbolo de igual (=) entre `nomeCampo` e `valorCampo`, sem espaço em branco. Campos com valores nulos devem ser identificados com NULO. Adicionalmente, os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

Para cada registro recuperado do arquivo de dados **pessoa**, gere a seguinte saída.
"Dados da pessoa de codigo " escrever o valor de idPessoa
"Nome: " escrever o valor de nomePessoa
"Idade: " escrever o valor de idadePessoa
"Usuario: " escrever o valor de nomeUsuario
Deixe uma linha em branco
Enquanto existirem registros do arquivo de dados **segueOrdenado**
tal que idPessoa = idPessoaQueSegue, gere a seguinte saída:
"Segue a pessoa de codigo: " escrever valor de idPessoaQueESeguida
"Justificativa para seguir: " escrever "celebridade" quando
grauAmizade = '0', escrever "amiga de minha amiga" quando
grauAmizade = '1' ou escrever "minha amiga" quando
grauAmizade = '2'
"Começou a seguir em: " escrever o valor de dataInicioQueSegue
"Parou de seguir em: " escrever o valor de dataFimQueSegue
Deixe uma linha em branco
Fim enquanto
Valores nulos devem substituídos pelo caractere "-".

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
10 pessoa.bin indexaPessoa.bin segueOrdenado.bin 1
1 idPessoa=25
Dados da pessoa de codigo 25
Nome: SAMANTHA PEREIRA CAMPOS
Idade: 13
Usuario: SAMANTHAPS
linha em branco
Segue a pessoa de codigo: 45
Justificativa para seguir: celebridade
Começou a seguir em: 01/01/2020
```

```
Parou de seguir em: 10/05/2025
linha em branco
Segue a pessoa de código: 29
Justificativa para seguir: minha amiga
Começou a seguir em: 03/05/2023
Parou de seguir em: 31/12/2024
linha em branco
...
```

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.

Vídeo gravado.

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

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

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

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

Instruções de entrega.

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

- página: https://runcodes.icmc.usp.br/
- Código de matrícula: **3YE6**

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. Vídeos corrompidos ou que não puderem ser corretamente acessados receberão nota zero.

Critério de Correção

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

- Corretude da execução do programa.
- Atendimento às especificações do registro de cabeçalho e dos registros de dados.
- Atendimento às especificações da sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade.
- Qualidade da documentação entregue. A documentação interna terá um peso considerável no trabalho.
- Vídeo. Integrantes que não participarem da apresentação receberão nota 0 no trabalho correspondente.

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

Restrições adicionais sobre o critério de correção.

- O uso excessivo de ferramentas de inteligência artificial acarretará a atribuição de nota zero (0) ao trabalho.
- A não execução de um programa devido a erros de compilação implica que a nota final da parte do trabalho será igual a zero (0).
- O não atendimento às especificações do registro de cabeçalho e dos registros de dados implica que haverá uma diminuição expressiva na nota do trabalho.

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

Bom Trabalho!