Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Organização de Arquivos (SCC0215)

Docente
Profa. Dra. Cristina Dutra de Aguiar Ciferri
cdac@icmc.usp.br

Colaborador e monitor (Turma 1 – segunda-feira)
João Paulo Clarindo
jpcsantos@usp.br
Luis Eduardo Rozante de Freitas Pereira
luis.eduardo.rozante@usp.br ou telegram: @LuisEduardo_Cabral

Monitores (Turma 2 – terça-feira)
Mateus Prado Santos
mateus.prado@usp.br
Lucas de Medeiros Franca Romero
lucasromero@usp.br ou telegram: @lucasromero

Terceiro Trabalho Prático

Este trabalho tem como objetivo realizar a junção entre dois arquivos de dados, com e sem o uso de índices.

O trabalho deve ser feito por, no máximo, 2 alunos da mesma turma. Os alunos devem ser os mesmos alunos do primeiro e do segundo trabalhos práticos. Caso haja mudanças, elas devem ser informadas para a docente e os monitores. 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 Organização de Arquivos é uma disciplina fundamental para a disciplina de Bases de Dados. A definição dos trabalhos práticos será 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 dos trabalhos práticos, os alunos poderão entrar em contato desde a disciplina de Organização de Arquivos com alguns comandos da linguagem SQL e verificar como eles são implementados.

Este trabalho prático tem como objetivo implementar a operação de junção. Ela é amplamente utilizada em aplicações de banco de dados desde que ela "junta" dados de registros de dois arquivos usando como base um campo de igualdade (condição de junção).

Considere um primeiro arquivo de dados `arquivoA` com os campos `Acampo1`, `Acampo2`, ..., `AcampoN`. Considere um segundo arquivo de dados `arquivoB` com os campos `Bcampo1`, `Bcampo2`, ..., `BcampoM`. As seguintes situações podem ocorrer na junção de `arquivoA` e `arquivoB` considerando a condição de junção `Acampo1 = Bcampo1`:

- Não existe igualdade entre `Acampo1` e `Bcampo1`. Nesse caso, nenhum registro é gerado como resultado da junção.
- Existe apenas uma igualdade entre `Acampo1` e `Bcampo1`. Nesse caso, somente um registro é gerado como resultado da junção.
- Existem *k* igualdades entre `Acampo1` e `Bcampo1`. Nesse caso, são gerados *k* registros como resultado da junção.

Programa

Descrição Geral. Implemente um programa em C por meio do qual seja possível realizar a junção de dois arquivos de dados, considerando diferentes formas de implementação dessa operação.

Importante. A definição da sintaxe de cada comando bem como sua saída devem seguir estritamente as especificações definidas em cada funcionalidade. Para especificar a sintaxe de execução, considere que o programa seja chamado de "programaTrab". Essas orientações devem ser seguidas uma vez que a correção do funcionamento do programa se dará de forma automática. De forma geral, a primeira entrada da entrada padrão é sempre o identificador de suas funcionalidades, conforme especificado a seguir.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

Na linguagem SQL, o comando SELECT é usado para listar os dados de uma ou mais tabelas (arquivos de dados). Quando mais do que uma tabela é usada, essas tabelas são "juntadas" por meio de uma (ou mais) coluna(s) de junção. Existem duas diferentes formas de se especificar a junção no comando SELECT:

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela1, tabela2 (ou seja, arquivos que contêm os campos)
(podem existir mais tabelas)
WHERE tabela1.nomeCampo = tabela2.nomeCampo (condição de junção)
AND critério de seleção (ou seja, critério de busca)

ou

SELECT lista de colunas (ou seja, campos a serem exibidos na resposta)
FROM tabela1 JOIN tabela2 ON tabela1.nomeCampo = tabela2.nomeCampo
(ou seja, arquivos que contêm os campos + condição de junção)
(podem existir mais tabelas)
WHERE critério de seleção (ou seja, critério de busca)

[15] Permita a recuperação dos dados de todos os registros armazenados no arquivo de dados **veiculo.bin**, juntando-os de forma apropriada com os dados de **linha.bin**. Essa funcionalidade requer a realização de uma junção entre **veiculo.bin** e **linha.bin**, considerando como condição de junção **veiculo.codLinha = linha.codLinha**.

Existem várias formas de se implementar a junção e, nesta funcionalidade, ela deve ser implementada por meio da **junção de loop aninhado**. A junção de loop aninhado é um algoritmo padrão (de força bruta), desde que não exige o uso de índices ou outras melhorias em qualquer arquivo para realizar a junção.

O algoritmo da **junção de loop aninhado** é definido como segue. Para cada registro presente no arquivo `arquivoA` (loop externo), recupere cada registro do arquivo `arquivoB` (loop interno) e teste se os dois registros satisfazem à condição de junção `Acampo1 = Bcampo1`. Ou seja:

para cada registro em `veiculo` faça
  para cada registro em `linha` faça
    se `veiculo.codLinha` = `linha.codLinha`
      então mostre os campos de `veiculo` e `linha` conforme solicitado
    fim-se
  fim-para
fim-para

Entrada do programa para a funcionalidade [15]:
`15 veiculo.bin linha.bin nomeCampoVeiculo nomeCampoLinha`
onde:
- `veiculo.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **veiculo** do primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos**.
- `linha.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **linha** do primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos**.
- `nomeCampoVeiculo` é o nome do campo do arquivo de dados **veiculo** que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **codLinha** pode ser utilizado.
- `nomeCampoLinha` é o nome do campo do arquivo de dados **linha** que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **codLinha** pode ser utilizado.

Mensagem de saída caso algum erro seja encontrado:
`Falha no processamento do arquivo.`

Mensagem de saída caso não seja gerado nenhum registro na junção dos dois arquivos:
`Registro inexistente.`

Saída caso o programa seja executado com sucesso:
`A saída deve ser exibida da seguinte forma. Para cada registro do arquivo veiculo.bin, primeiro mostre os seus campos. Depois, mostre os campos do registro do arquivo linha.bin correspondente.`

`Com relação aos campos do arquivo veiculo.bin, eles devem ser mostrados da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por "campo com valor nulo". A data deve ser exibida na forma corrente de escrita.`

`Com relação aos campos do arquivo linha.bin, eles devem ser mostrados da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por "campo com valor nulo". Com relação ao campo aceita cartão, seu valor deve ser exibido como PAGAMENTO SOMENTE COM CARTAO SEM PRESENCA DE COBRADOR (ao invés de S) ou PAGAMENTO EM CARTAO E DINHEIRO (ao invés de N) ou PAGAMENTO EM CARTAO SOMENTE NO FINAL DE SEMANA (ao invés de F).`

Depois de exibidos todos os campos de um veículo e de sua respectiva linha, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada no **exemplo de execução.**

Exemplo de execução:

```
./programaTrab
15 veiculo.bin linha.bin codLinha codLinha

Prefixo do veiculo: BI854
Modelo do veiculo: MARCOLOPO TORINO
Categoria do veiculo: MICROESPECIAL
Data de entrada do veiculo na frota: 28 de janeiro de 2019
Quantidade de lugares sentados disponiveis: 16
Codigo da linha: 150
Nome da linha: C. MUSICA-V.ALEGRE
Cor que descreve a linha: AMARELA
Aceita cartao: PAGAMENTO SOMENTE COM CARTAO SEM PRESENCA DE COBRADOR
--- pule uma linha em branco ---
Prefixo do veiculo: BL608
Modelo do veiculo: MARCOLOPO TORINO
Categoria do veiculo: ARTICULADO
Data de entrada do veiculo na frota: 24 de abril de 2019
Quantidade de lugares sentados disponiveis: campo com valor nulo
Codigo da linha: 23
Nome da linha: INTER 2 (ANTI-HORARIO)
Cor que descreve a linha: PRATA
Aceita cartao: PAGAMENTO EM CARTAO E DINHEIRO
--- pule uma linha em branco ---
```

[16] Permita a recuperação dos dados de todos os registros armazenados no arquivo de dados **veiculo.bin**, juntando-os de forma apropriada com os dados de **linha.bin**. Essa funcionalidade requer a realização de uma junção entre **veiculo.bin** e **linha.bin**, considerando como condição de junção **veiculo.codLinha = linha.codLinha**. Existem várias formas de se implementar a junção e, nesta funcionalidade, ela deve ser implementada por meio da **junção de loop único**. A junção de loop único depende da existência de índices. Portanto, o índice criado no trabalho prático 2 deve ser usado neste trabalho.

O algoritmo da **junção de loop único** é definido como segue. Para cada registro presente no arquivo `arquivoA` (loop externo), recupere cada registro do arquivo `arquivoB` (loop interno) usando o índice e teste se os dois registros satisfazem à condição de junção `Acampo1 = Bcampo1`. Ou seja:

para cada registro em `veiculo` faça
    selecione os registros de `linha` que satisfaçam à condição
        `linha.codLinha` = `veiculo.codLinha` usando o índice árvore-B definido sobre o arquivo `linha`
    mostre os campos de `veiculo` e `linha` conforme solicitado
fim-para

Entrada do programa para a funcionalidade [16]:
`16 veiculo.bin linha.bin nomeCampoVeiculo nomeCampoLinha indiceLinha`
onde:
- `veiculo.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **veiculo** do primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos**.
- `linha.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **linha** do primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos**.
- `nomeCampoVeiculo` é o nome do campo do arquivo de dados **veiculo** que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **codLinha** pode ser utilizado.
- `nomeCampoLinha` é o nome do campo do arquivo de dados **linha** que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **codLinha** pode ser utilizado.
- `indiceLinha` é o índice árvore-B definido sobre o arquivo de dados **linha** do segundo trabalho prático.

Mensagem de saída caso algum erro seja encontrado:
`Falha no processamento do arquivo.`

Mensagem de saída caso não seja gerado nenhum registro na junção dos dois arquivos:
`Registro inexistente.`

Saída caso o programa seja executado com sucesso:
`A saída deve ser exibida da seguinte forma. Para cada registro do arquivo veiculo.bin, primeiro mostre os seus campos. Depois, mostre os campos do registro do arquivo linha.bin correspondente.`

`Com relação aos campos do arquivo linha.bin, eles devem ser mostrados da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por "campo com valor nulo". A data deve ser exibida na forma corrente de escrita.`

`Com relação aos campos do arquivo linha.bin, eles devem ser mostrados da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por "campo com valor nulo". Com relação ao campo aceita cartão, seu valor deve ser exibido como PAGAMENTO SOMENTE COM CARTAO SEM PRESENCA DE COBRADOR (ao invés de S) ou PAGAMENTO EM CARTAO E DINHEIRO (ao invés`

de N) ou PAGAMENTO EM CARTAO SOMENTE NO FINAL DE SEMANA (ao invés de F).

Depois de exibidos todos os campos de um veículo e de sua respectiva linha, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada no **exemplo de execução.**

Exemplo de execução:

```
./programaTrab
16 veiculo.bin linha.bin codLinha codLinha indiceArvoreBsobreLinha

Prefixo do veiculo: BI854
Modelo do veiculo: MARCOLOPO TORINO
Categoria do veiculo: MICROESPECIAL
Data de entrada do veiculo na frota: 28 de janeiro de 2019
Quantidade de lugares sentados disponiveis: 16
Codigo da linha: 150
Nome da linha: C. MUSICA-V.ALEGRE
Cor que descreve a linha: AMARELA
Aceita cartao: PAGAMENTO SOMENTE COM CARTAO SEM PRESENCA DE COBRADOR
--- pule uma linha em branco ---
Prefixo do veiculo: BL608
Modelo do veiculo: MARCOLOPO TORINO
Categoria do veiculo: ARTICULADO
Data de entrada do veiculo na frota: 24 de abril de 2019
Quantidade de lugares sentados disponiveis: campo com valor nulo
Codigo da linha: 23
Nome da linha: INTER 2 (ANTI-HORARIO)
Cor que descreve a linha: PRATA
Aceita cartao: PAGAMENTO EM CARTAO E DINHEIRO
--- pule uma linha em branco ---
```

[17] Ordene o arquivo de dados **veiculo.bin** de acordo com um campo de ordenação **codLinha**. A ordenação deve ser implementada considerando que o arquivo de dados cabe totalmente em memória primária (RAM). Portanto, o arquivo deve ser: (i) lido inteiramente do disco para a RAM; (ii) ordenado 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, o qual encontra-se ordenado de acordo com o campo de ordenação. O arquivo de dados original não deve ser removido. Registros marcados como logicamente removidos não devem estar presentes no arquivo ordenado gerado pela funcionalidade.

Entrada do programa para a funcionalidade [17]:

`17 arquivoDesordenado.bin arquivoOrdenado.bin campoOrdenacao`

onde:

- `arquivoDesordenado.bin` é o arquivo binário de entrada **veiculo.bin** que segue as mesmas especificações do arquivo de dados definido no primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos.**
- `arquivoOrdenado.bin` é um arquivo binário de saída que segue as mesmas especificações do arquivo de dados definido no primeiro trabalho prático, e que contém dados **ordenados** e **não contém registros logicamente removidos.**
- `campoOrdenacao` é um campo do arquivo que é utilizado como base para a ordenação. No presente trabalho, é o campo `codLinha`.

Saída caso o programa seja executado com sucesso:

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

Mensagem de saída caso algum erro seja encontrado:

`Falha no carregamento do arquivo.`

Exemplo de execução:

```
./programaTrab
17 veiculo.bin veiculoOrdenado.bin codLinha
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo binário veiculoOrdenado
```

[18] Ordene o arquivo de dados **linha.bin** de acordo com um campo de ordenação **codLinha**. A ordenação deve ser implementada considerando que o arquivo de dados cabe totalmente em memória primária (RAM). Portanto, o arquivo deve ser: (i) lido inteiramente do disco para a RAM; (ii) ordenado 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, o qual encontra-se ordenado de acordo com o campo de ordenação. O arquivo de dados original não deve ser removido. Registros marcados como logicamente removidos não devem estar presentes no arquivo ordenado gerado pela funcionalidade.

Entrada do programa para a funcionalidade [18]:

`18 arquivoDesordenado.bin arquivoOrdenado.bin campoOrdenacao`

onde:

- `arquivoDesordenado.bin` é um arquivo binário de entrada `linha.bin` que segue as mesmas especificações do arquivo de dados definido no primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos**.
- `arquivoOrdenado.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados definido no primeiro trabalho prático, e que contém dados **ordenados** e **não contém registros logicamente removidos**.
- `campoOrdenacao` é um campo do arquivo que é utilizado como base para a ordenação. No presente trabalho, é o campo `codLinha`.

Saída caso o programa seja executado com sucesso:

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

Mensagem de saída caso algum erro seja encontrado:

`Falha no carregamento do arquivo.`

Exemplo de execução:

```
./programaTrab
18 linha.bin linhaOrdenado.bin codLinha
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo binário linhaOrdenado
```

[19] Permita a recuperação dos dados de todos os registros armazenados no arquivo de dados **linha.bin**, juntando-os de forma apropriada com os dados de **veiculo.bin**. Essa funcionalidade requer a realização de uma junção entre **linha.bin** e **veiculo.bin**, considerando como condição de junção **linha.codLinha = veiculo.codLinha**. Existem várias formas de se implementar a junção e, nesta funcionalidade, ela deve ser implementada por meio da **junção ordenação-intercalação**. A junção ordenação-intercalação é possível de ser utilizada quando os registros presentes em ambos `arquivoA` e `arquivoB` estão ordenados pelo valor dos atributos da condição de junção, ou seja, `Acampo1` e `Bcampo1`, respectivamente.

O algoritmo da **junção ordenação-intercalação** é definido como segue. Ordene cada arquivo de dados com base no atributo da condição de junção. Percorra os dois arquivos simultaneamente na ordem dos atributos de junção, combinando os registros que têm os mesmos valores para a condição de junção. Quando os arquivos encontram-se ordenados, esse é o algoritmo mais eficiente. Ou seja:

ordene o arquivo `veiculo` considerando o campo `codLinha`
ordene o arquivo `linha` considerando o campo `codLinha`
realize o **merge** dos arquivos veiculo e linha ordenados
mostre os campos de `veiculo` e `linha` conforme solicitado

Entrada do programa para a funcionalidade [19]:
`19 veiculo.bin linha.bin nomeCampoVeiculo nomeCampoLinha`
onde:

- `veiculoDesordenado.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **veiculo** definido no primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos**. Antes de se realizar o merge dos arquivos, `veiculoDesordenado.bin` deve ser ordenado usando a funcionalidade [17] com `nomeCampoVeiculo` como campo de ordenação.
- `linhaDesordenado.bin` é um arquivo binário de entrada que segue as mesmas especificações do arquivo de dados **linha** definido no primeiro trabalho prático, e que contém dados **desordenados** e **registros logicamente removidos.** Antes de se realizar o merge dos arquivos, `linhaDesordenado.bin` deve ser ordenado usando a funcionalidade [18] com `nomeCampoLinha` como campo de ordenação.
- `nomeCampoVeiculo` é o nome do campo do arquivo de dados **veiculo** que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **codLinha** pode ser utilizado.
- `nomeCampoLinha` é o nome do campo do arquivo de dados **linha** que está sendo usado como condição de junção. Neste trabalho prático, apenas o campo **codLinha** pode ser utilizado.

Mensagem de saída caso algum erro seja encontrado:
`Falha no processamento do arquivo.`

Mensagem de saída caso não seja gerado nenhum registro na junção dos dois arquivos:
`Registro inexistente.`

Saída caso o programa seja executado com sucesso:
`A saída deve ser exibida da seguinte forma. Para cada registro do arquivo veiculo.bin, primeiro mostre os seus campos. Depois, mostre os campos do registro do arquivo linha.bin correspondente.`

`Com relação aos campos do arquivo linha.bin, eles devem ser mostrados da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por "campo com valor nulo". A data deve ser exibida na forma corrente de escrita.`

Com relação aos campos do arquivo linha.bin, eles devem ser mostrados da seguinte forma. Cada campo deve ser exibido em uma linha diferente. Primeiro, deve ser colocado o valor do metadado para aquele campo, e depois o seu valor. Campos com valores nulos devem ser representados por "campo com valor nulo". Com relação ao campo aceita cartão, seu valor deve ser exibido como PAGAMENTO SOMENTE COM CARTAO SEM PRESENCA DE COBRADOR (ao invés de S) ou PAGAMENTO EM CARTAO E DINHEIRO (ao invés de N) ou PAGAMENTO EM CARTAO SOMENTE NO FINAL DE SEMANA (ao invés de F).

Depois de exibidos todos os campos de um veículo e de sua respectiva linha, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada no **exemplo de execução.**

Exemplo de execução:

```
./programaTrab
18 veiculo.bin linha.bin codLinha codLinha

Prefixo do veiculo: BI854
Modelo do veiculo: MARCOLOPO TORINO
Categoria do veiculo: MICROESPECIAL
Data de entrada do veiculo na frota: 28 de janeiro de 2019
Quantidade de lugares sentados disponiveis: 16
Codigo da linha: 150
Nome da linha: C. MUSICA-V.ALEGRE
Cor que descreve a linha: AMARELA
Aceita cartao: PAGAMENTO SOMENTE COM CARTAO SEM PRESENCA DE COBRADOR
--- pule uma linha em branco ---
Prefixo do veiculo: BL608
Modelo do veiculo: MARCOLOPO TORINO
Categoria do veiculo: ARTICULADO
Data de entrada do veiculo na frota: 24 de abril de 2019
Quantidade de lugares sentados disponiveis: campo com valor nulo
Codigo da linha: 23
Nome da linha: INTER 2 (ANTI-HORARIO)
Cor que descreve a linha: PRATA
Aceita cartao: PAGAMENTO EM CARTAO E DINHEIRO
--- pule uma 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.
- 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
- **Turma 1** (segunda-feira): código de matrícula: **7Y6N**
- **Turma 2** (terça-feira): código de matrícula: **H6A3**

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.
