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

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 recuperar todos os dados armazenados.

O trabalho deve ser feito por, no máximo, 2 alunos da mesma turma. A solução deve ser proposta exclusivamente pelo(s) aluno(s) com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.

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.

Os trabalhos práticos têm como objetivo armazenar e recuperar dados relacionados à frota de ônibus da cidade de Curitiba, Paraná (http://dadosabertos.c3sl.ufpr.br/curitibaurbs/). Esses dados se referem aos veículos e às linhas às quais esses veículos estão associados. Um veículo pode estar associado a apenas uma única linha, enquanto uma linha pode ter vários veículos associados a ela. Na disciplina de Bases de Dados, é ensinado que o projeto deve ser feito em dois arquivos. O primeiro arquivo armazena dados de veículos e um código que identifica a linha à qual cada veículo está associado. O segundo arquivo armazena dados relacionados a linhas.

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 **veiculo** e arquivo de dados **linha**.

Descrição do arquivo de dados veiculo

O arquivo de dados **veiculo** 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.
- *byteProxReg*: *byte offset* do próximo registro a ser inserido. Ou seja, *byte offset* no qual um novo registro deve ser inserido – inteiro – tamanho: 8 bytes

- *nroRegistros*: número de registros presentes no arquivo de dados – inteiro – tamanho: 4 bytes
- *nroRegRemovidos*: número de registros removidos presentes no arquivo de dados – inteiro – tamanho: 4 bytes
- *descrevePrefixo*: descrição completa do campo *prefixo*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 18 bytes.
- *descreveData*: descrição completa do campo *data*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 35 bytes.
- *descreveLugares*: descrição completa do campo *quantidadeLugares*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 42 bytes.
- *descreveLinha*: descrição completa do campo *codLinha*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 26 bytes.
- *descreveModelo*: descrição completa do campo *modelo*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 17 bytes.
- *descreveCategoria*: descrição completa do campo *categoria*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 20 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 174 bytes, representado da seguinte forma:

1 byte	8 bytes	4 bytes	4 bytes	18 bytes	35 bytes	42 bytes	17 bytes	20 bytes	26 bytes
status	*byteProxReg*	*nro Registros*	*nroReg Removidos*	*descreve Prefixo*	*descreve Data*	*descreve Lugares*	*descreve Linha*	*descreve Modelo*	*descreve Categoria*
0	1…8	9…12	13…16	17…34	35…69	70…111	112…128	129…148	149…174

Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Todos os campos são *strings* de tamanho fixo e foram devidamente projetados de acordo com os dados do arquivo .csv. 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 registros e campos de tamanho variável, deve ser usado o método indicador de tamanho. Os campos são definidos da seguinte forma:

- *prefixo*: código que identifica univocamente cada registro do arquivo de dados – *string* de tamanho fixo de 5 bytes.
- *data*: data de entrada do veículo na frota – *string* de tamanho fixo de 10 bytes, no formato AAAA-MM-DD
- *quantidadeLugares*: quantidade de lugares sentados disponíveis – inteiro – tamanho: 4 bytes.
- *codLinha*: linha associada ao veículo – inteiro – tamanho: 4 bytes.
- *modelo:* modelo do veículo – *string* de tamanho variável.
- *categoria*: categoria do veículo – *string* de tamanho variável.

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

- *removido*: indica se o registro está logicamente removido. Pode assumir os valores '0', para indicar que o registro está marcado como logicamente removido, ou '1', 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.
- *tamanhoModelo:* número de bytes do campo *modelo* – inteiro – tamanho: 4 bytes.
- *tamanhoCategoria:* número de bytes do campo *categoria* – inteiro – tamanho: 4 bytes.

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

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

1 byte	4 bytes	5 bytes	10 bytes	4 bytes	4 bytes	4 bytes	variável	4 bytes	variável
removido	*tamanho Registro*	*prefixo*	*data*	*quantidade Lugares*	*cod Linha*	*tamanho Modelo*	*modelo*	*tamanho Categoria*	*categoria*
0	1...4	5...9	10...19	20...23	24...27	28...31	32...	...	...

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Quando necessário, as *strings* devem ser finalizadas com '\0' e o lixo deve ser identificado pelo caractere '@'. Nenhum *byte* do registro deve permanecer vazio, ou seja, cada *byte* deve armazenar um valor válido ou '\0' ou '@'.
- O campo *prefixo* não aceita 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 da seguinte forma:
    - se o campo é inteiro, então armazena-se o valor -1.
    - se o campo é do tipo *string*, então armazena-se '\0@@@@', por exemplo.
- Para os campos de tamanho variável, os valores nulos devem ser represntados da seguinte forma:
    - deve ser armazenado o valor 0 no tamanho do campo.
    - não deve ser armazenado nada no campo correspondente.
- 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.

Descrição do arquivo de dados linha

O arquivo de dados **linha** 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.
- *byteProxReg*: *byte offset* do próximo registro a ser inserido. Ou seja, *byte offset* no qual um novo registro deve ser inserido – inteiro – tamanho: 8 bytes
- *nroRegistros*: número de registros presentes no arquivo de dados – inteiro – tamanho: 4 bytes
- *nroRegRemovidos*: número de registros removidos presentes no arquivo de dados – inteiro – tamanho: 4 bytes
- *descreveCodigo*: descrição completa do campo *codLinha*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 15 bytes.
- *descreveCartao*: descrição completa do campo *aceitaCartao*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 13 bytes.
- *descreveNome*: descrição completa do campo *nomeLinha*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 13 bytes.
- *descreveCor*: descrição completa do campo *corLinha*. Deve assumir o valor que foi lido do arquivo .csv – tamanho: *string* de tamanho fixo de 24 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 82 bytes, representado da seguinte forma:

1 byte	8 bytes	4 bytes	4 bytes	15 bytes	13 bytes	13 bytes	24 bytes
status	*byteProxReg*	*nro Registros*	*nroReg Removidos*	*descreve Código*	*descreve Cartao*	*descreve Nome*	*descreve Linha*
0	1…8	9…12	13…16	17…31	32…44	45…57	58…82

Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Todos os campos são *strings* de tamanho fixo e foram devidamente projetados de acordo com os dados do arquivo .csv. 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 registros e campos de tamanho variável, deve ser usado o método indicador de tamanho. Os campos são definidos da seguinte forma:

- *codLinha*: código que identifica univocamente cada registro do arquivo de dados – inteiro – tamanho: 4 bytes.
- *aceitaCartao*: indica se a linha aceita somente cartão. Pode assumir os valores: (i) 'S' para indicar que a linha não possui cobrador; (ii) 'N' para indicar que, além do cartão, a linha também aceita dinheiro; (iii) 'F' para indicar que, nos finais de semana, a linha somente aceita cartão como forma de pagamento – tamanho: *string* de 1 byte
- *nomeLinha:* nome da linha – *string* de tamanho variável.
- *corLinha*: cor que identifica a linha – *string* de tamanho variável.

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

- *removido*: indica se o registro está logicamente removido. Pode assumir os valores '0', para indicar que o registro está marcado como logicamente removido, ou '1', 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.
- *tamanhoNome:* número de bytes do campo *nomeLinha* – inteiro – tamanho: 4 bytes.
- *tamanhoCor:* número de bytes do campo *corLinha* – inteiro – tamanho: 4 bytes.

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

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

1 byte	4 bytes	4 bytes	1 byte	4 bytes	variável	4 bytes	variável
removido	*tamanho Registro*	*codLinha*	*aceitaCartao*	*tamanho Nome*	*nomeLinha*	*tamanhoCor*	*corLinha*
0	1...4	5...8	9	10...13	14...	...	...

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Quando necessário, as *strings* devem ser finalizadas com '\0' e o lixo deve ser identificado pelo caractere '@'. Nenhum *byte* do registro deve permanecer vazio, ou seja, cada *byte* deve armazenar um valor válido ou '\0' ou '@'.
- O campo *codLinha* não aceita 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 da seguinte forma:
    - se o campo é inteiro, então armazena-se o valor -1.

- se o campo é do tipo *string*, então armazena-se '\0@@@@@', por exemplo.

- Para os campos de tamanho variável, os valores nulos devem ser represntados da seguinte forma:
  - deve ser armazenado o valor 0 no tamanho do campo.
  - não deve ser armazenado nada no campo correspondente.
- 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 listar todos os 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. As funcionalidades [1] e [2] representam exemplos de implementação do comando CREATE TABLE.

[1] Permita a leitura de vários registros obtidos a partir do arquivo de entrada **veiculo.csv** e a gravação desses registros em um arquivo de dados de saída. O arquivo de entrada **veiculo.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. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo binário. Lembre-se de manipular o campo "status" do registro de cabeçalho adequadamente. Primeiro ele deve começar com o valor '0' (arquivo inconsistente) e, só ao final da execução do programa e ao final de todas as gravações, ele deve mudar para o valor '1' (arquivo consistente). Lembre-se também de manipular o campo "removido" dos registros de dados adequadamente. Quando um registro for marcado como removido no arquivo de entrada **veiculo.csv**, o valor do campo "removido" desse registro deve assumir o valor '0'. No arquivo **veiculo.csv**, registros removidos começam com o caractere '*'.

Entrada do programa para a funcionalidade [1]:

```
1 veiculo.csv veiculo.bin
```

onde:

- `veiculo.csv` é um arquivo .csv que contém os valores dos campos dos registros a serem armazenados no arquivo binário.
- `veiculo.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 veiculo.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 veiculo.csv veiculo.bin
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo veiculo.bin
```

[2] Permita a leitura de vários registros obtidos a partir do arquivo de entrada **linha.csv** e a gravação desses registros em um arquivo de dados de saída. O arquivo de entrada **linha.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. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo binário. Lembre-se de manipular o campo "status" do registro de cabeçalho adequadamente. Primeiro ele deve começar com o valor '0' (arquivo inconsistente) e, só ao final da execução do programa e ao final de todas as gravações, ele deve mudar para o valor '1' (arquivo consistente). Lembre-se também de manipular o campo "removido" dos registros de dados adequadamente. Quando um registro for marcado como removido no arquivo de entrada **linha.csv**, o valor do campo "removido" desse registro deve assumir o valor '0'. No arquivo **linha.csv**, registros removidos começam com o caractere '*'.

Entrada do programa para a funcionalidade [2]:

```
2 linha.csv linha.bin
```

onde:

- `linha.csv` é um arquivo .csv que contém os valores dos campos dos registros a serem armazenados no arquivo binário.
- `linha.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 veiculo.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
2 linha.csv linha.bin
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo linha.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)

As funcionalidades [3] e [4] representam exemplos de implementação do comando SELECT.

[3] Permita a recuperação dos dados de todos os registros armazenados no arquivo de dados **veiculo.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 [3]:

`3 veiculo.bin`

onde:

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

Saída caso o programa seja executado com sucesso:

`Para cada registro, devem ser exibidos seus campos 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. Depois de cada registro, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada 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
3 veiculo.bin
Prefixo do veiculo: BA004
Modelo do veiculo: NEOBUS MEGA
Categoria do veiculo: ALIMENTADOR
Data de entrada do veiculo na frota: 29 de maio de 2009
Quantidade de lugares sentados disponiveis: campo com valor nulo
--- pule uma linha em branco ---
```

[4] Permita a recuperação dos dados de todos os registros armazenados no arquivo de dados **linha.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 [4]:

```
4 linha.bin
```

onde:

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

Saída caso o programa seja executado com sucesso:

Para cada registro, devem ser exibidos seus campos 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 cada registro, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada 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
4 linha.bin
Codigo da linha: 464
Nome da linha: campo com valor nulo
Cor que descreve a linha: AMARELA
Aceita cartao: PAGAMENTO SOMENTE COM CARTAO SEM PRESENCA DE COBRADOR
--- pule uma linha em branco ---
```

Conforme visto nas funcionalidades [3] e [4], 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)

As funcionalidades [5] e [6] representam exemplos de implementação do comando SELECT considerando a cláusula WHERE.

[5] Permita a recuperação dos dados de todos os registros do arquivo **veiculo.bin** que satisfaçam um critério de busca determinado pelo usuário. Qualquer campo pode ser utilizado como forma de busca. Essa 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. A busca deve ser feita considerando apenas um único campo. Registros marcados como logicamente removidos não devem ser exibidos.

Sintaxe do comando para a funcionalidade [5]:

`5 veiculo.bin NomeDoCampo valor`

onde:

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

- `nomeDoCampo` é um dos seguintes nomes de campos de `veiculo.bin`: `prefixo, data, quantidadeLugares, modelo, categoria.`

- `valor` é o valor do campo definido em `nomeDoCampo` que deve ser passado como parâmetro para a busca.

Saída caso o programa seja executado com sucesso:

`Para cada registro, devem ser exibidos seus campos 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. Depois de cada registro, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada 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
5 veiculo.bin prefixo "BA004"
Prefixo do veiculo: BA004
Modelo do veiculo: NEOBUS MEGA
Categoria do veiculo: ALIMENTADOR
Data de entrada do veiculo na frota: 29 de maio de 2009
Quantidade de lugares sentados disponiveis: campo com valor nulo
--- pule uma linha em branco ---
```

[6] Permita a recuperação dos dados de todos os registros do arquivo **linha.bin** que satisfaçam um critério de busca determinado pelo usuário. Qualquer campo pode ser utilizado como forma de busca. Essa 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. A busca deve ser feita considerando apenas um único campo. Registros marcados como logicamente removidos não devem ser exibidos.

Sintaxe do comando para a funcionalidade [6]:

```
6 linha.bin NomeDoCampo valor
```

onde:

- `linha.bin` é o arquivo binário gerado conforme as especificações descritas neste trabalho prático.
- `nomeDoCampo` é um dos seguintes nomes de campos de `linha.bin`: `codLinha, aceitaCartao, nomeLinha, corLinha.`
- `valor` é o valor do campo definido em `nomeDoCampo` que deve ser passado como parâmetro para a busca.

Saída caso o programa seja executado com sucesso:

Para cada registro, devem ser exibidos seus campos 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 cada registro, pule uma linha em branco. A ordem de exibição dos campos dos registros é ilustrada 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
6 linha.bin codLinha 464
Nome da linha: campo com valor nulo
Cor que descreve a linha: AMARELA
Aceita cartao: PAGAMENTO EM CARTAO SEM COBRADOR
--- pule uma linha em branco ---
```

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. As funcionalidades [7] e [8] representam exemplos de implementação do comando INSERT INTO.

[7] Permita a inserção de novos registros no arquivo de entrada **veiculo.bin**. Neste projeto, a inserção dos registros deve ser feita sempre no final do arquivo de dados. O novo registro não deve ser marcado como removido. 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 [7] 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 [7]:

`7 veiculo.bin n`

$\texttt{prefixo}_1$ $\texttt{data}_1$ $\texttt{quantidadeLugares}_1$ $\texttt{codLinha}_1$ $\texttt{modelo}_1$ $\texttt{categoria}_1$

$\texttt{prefixo}_2$ $\texttt{data}_2$ $\texttt{quantidadeLugares}_2$ $\texttt{codLinha}_2$ $\texttt{modelo}_2$ $\texttt{categoria}_2$

...

$\texttt{prefixo}_n$ $\texttt{data}_n$ $\texttt{quantidadeLugares}_n$ $\texttt{codLinha}_n$ $\texttt{modelo}_n$ $\texttt{categoria}_n$

onde:

- `veiculo.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, para os campos especificados na mesma ordem que a definida nesse trabalho prático, a saber: `prefixo`, `data`, `quantidadeLugares`, `codLinha`, `modelo`, `categoria`. 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 veiculo.bin usando a função binarioNaTela.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
7 veiculo.bin 2
“AT090“ “2019-05-20“ 30 333 NULO “VERMELHO“
“XX997“ “2015-09-30“  20 672 NULO NULO
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo veiculo.bin, o qual foi atualizado com as inserções.
```

[8] Permita a inserção de novos registros no arquivo de entrada **linha.bin**. Neste projeto, a inserção dos registros deve ser feita sempre no final do arquivo de dados. 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. Na entrada da funcionalidade, quando prefixo começar com o caractere '*', o novo registro deve ser marcado logicamente como removido. Caso contrário, ou seja, quando o prefixo não começar com o caractere '*', o novo registro não deve ser marcado como removido. A funcionalidade [8] 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 [8]:

`8 linha.bin n`

$codLinha_1$ $aceitaCartao_1$ $nomeLinha_1$ $corLinha_1$

$codLinha_2$ $aceitaCartao_2$ $nomeLinha_2$ $corLinha_2$

...

$codLinha_n$ $aceitaCartao_n$ $nomeLinha_n$ $corLinha_n$

onde:

- `linha.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, para os campos especificados na mesma ordem que a definida nesse trabalho prático, a saber: `codLinha`, `aceitaCartao`, `nomeLinha`, `corLinha`. 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 linha.bin usando a função binarioNaTela.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
8 veiculo.bin 1
333 "S" NULO "VERMELHO"
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo linha.bin, o qual foi atualizado com as inserções.`

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. A entrega deve ser feita 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 carregado no drive compartilhado da disciplina. Será criado um novo diretório chamado Trabalho Prático T1 para esse fim. O vídeo deve ser nomeado como **Grupo X** (onde X representa o número do grupo).

Critério de Correção

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

- Corretude da execução do programa.
- Atendimento às especificações do registro de cabeçalho e dos registros de dados.

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

Restrições adicionais sobre o critério de correção.

- A não execução de um programa devido a erros de compilação implica que a nota final da parte do trabalho será igual a zero (0).
- O não atendimento às especificações do registro de cabeçalho e dos registros de dados implica que haverá uma diminuição expressiva na nota do trabalho.
- O não atendimento às especificações de sintaxe dos comandos de cada funcionalidade e do formato de saída da execução de cada funcionalidade implica que haverá uma diminuição expressiva na nota do trabalho.
- A ausência da documentação implica que haverá uma diminuição expressiva na nota do trabalho.
- A 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!