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
cdac@icmc.usp.br

Aluno PAE
João Paulo Clarindo
jpcsantos@usp.br

Monitores
Lucas de Medeiros Franca Romero
lucasromero@usp.br *ou telegram: @lucasromero*
Eduardo Souza Rocha
eduardos.rocha17@usp.br *ou telegram: @edwolt*

Primeiro Trabalho Prático

Este trabalho tem como objetivo armazenar dados em arquivos binários de acordo com organizações de campos e registros diferentes, bem como desenvolver funcionalidades para recuperar dados desses arquivos.

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 deste primeiro trabalho prático é feita considerando esse aspecto, ou seja, o projeto será especificado em termos de várias funcionalidades, e essas funcionalidades serão relacionadas com as funcionalidades da linguagem SQL (*Structured Query Language*), que é a linguagem utilizada por sistemas gerenciadores de banco de dados (SGBDs) relacionais. As características e o detalhamento de SQL serão aprendidos na disciplina de Bases de Dados. Porém, por meio do desenvolvimento deste trabalho prático, os alunos poderão entrar em contato com alguns comandos da linguagem SQL e verificar como eles são implementados.

Os trabalhos práticos têm como objetivo armazenar e recuperar dados relacionados à listagem da frota de veículos no Brasil, a qual é mantida pelo Denatran.

Arquivo Tipo 1
Descrição do Arquivo de Dados para Registros de Tamanho Fixo

Um arquivo de dados possui um registro de cabeçalho e 0 ou mais registros de dados. A descrição do registro de cabeçalho do arquivo Tipo 1 é feita conforme a definição a seguir.

Registro de Cabeçalho. O registro de cabeçalho deve conter os seguintes campos:

- *status*: indica a consistência do arquivo de dados, devido à queda de energia, travamento do programa, etc. Pode assumir os valores '0', para indicar que o arquivo de dados está inconsistente, ou '1', para indicar que o arquivo de dados está consistente. Ao 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.
- *topo:* armazena o RRN de um registro logicamente removido, ou -1 caso não haja registros logicamente removidos – tamanho: inteiro de 4 bytes.
- *descricao:* indica que será feita a descrição dos metadados do arquivo de dados, juntamente com sua descrição simplificada. Deve armazenar o valor 'LISTAGEM DA FROTA DOS VEICULOS NO BRASIL' – tamanho: *string* de 40 bytes
- desC1: especifica a descrição detalhada do primeiro campo. Deve armazenar o valor 'CODIGO IDENTIFICADOR: ' – tamanho: *string* de 15 bytes
- desC2: especifica a descrição detalhada do segundo campo. Deve armazenar o valor 'ANO DE FABRICACAO: ' – tamanho: *string* de 19 bytes
- desC3: especifica a descrição detalhada do terceiro campo. Deve armazenar o valor 'QUANTIDADE DE VEICULOS: ' – tamanho: *string* de 24 bytes
- desC4: especifica a descrição detalhada do quarto campo. Deve armazenar o valor 'ESTADO: ' – tamanho: *string* de 8 bytes

- codC5: especifica a descrição simplificada do quinto campo. Deve armazenar o valor '0' – tamanho: *string* de 1 byte
- desC5: especifica a descrição detalhada do quinto campo. Deve armazenar o valor 'NOME DA CIDADE: ' – tamanho: *string* de 17 bytes
- codC6: especifica a descrição simplificada do sexto campo. Deve armazenar o valor '1' – tamanho: *string* de 1 byte
- desC6: especifica a descrição detalhada do sexto campo. Deve armazenar o valor 'MARCA DO VEICULO: ' – tamanho: *string* de 18 bytes
- codC7: especifica a descrição simplificada do sétimo campo. Deve armazenar o valor '2' – tamanho: *string* de 1 byte
- desC7: especifica a descrição detalhada do sétimo campo. Deve armazenar o valor 'MODELO DO VEICULO: ' – tamanho: *string* de 19 bytes
- *proxRRN*: armazena o valor do próximo RRN disponível. Deve ser iniciado com o valor '0' e ser incrementado quando necessário – tamanho: inteiro de 4 bytes.
- *nroRegRem*: armazena o número de registros logicamente marcados como removidos. Deve ser iniciado com o valor '0' e deve ser incrementado sempre que necessário – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 182 bytes, representado da seguinte forma:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
status	topo				L	I	S	T	A	G	E	M		D	A		F	R	O
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
T	A		D	O	S		V	E	I	C	U	L	O	S		N	O		B
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
R	A	S	I	L	C	O	D	I	G	O		I	D	E	N	T	I	F	I
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
C	A	D	O	R	:		A	N	O		D	E		F	A	B	R	I	C
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
A	C	A	O	:		Q	U	A	N	T	I	D	A	D	E		D	E	
100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
V	E	I	C	U	L	O	S	:		E	S	T	A	D	O	:		0	N
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
O	M	E		D	A		C	I	D	A	D	E	:		1	M	A	R	C
140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A		D	O		V	E	I	C	U	L	O	:		2	M	O	D	E	L
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
O		D	O		V	E	I	C	U	L	O	:		proxRRN				nroRegRem	
180	181																		

Observações Importantes.

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

A descrição dos registros de dados do arquivo Tipo 1 é feita conforme a definição a seguir.

Registros de Dados. Os registros de dados são de tamanho fixo, com campos de tamanho fixo e campos de tamanho variável. Para os campos de tamanho variável, deve ser usado o método indicador de tamanho.

Os campos de tamanho fixo são definidos da seguinte forma:

- *id*: código identificador – inteiro – tamanho: 4 bytes.
- *ano*: ano de fabricação do veículo – inteiro – tamanho: 4 bytes.
- *qtt*: quantidade de veículos – inteiro – tamanho: 4 bytes.
- *sigla*: sigla do estado no qual os veículos estão cadastrados – string – tamanho: 2 bytes.

Os campos de tamanho variável são definidos da seguinte forma:

- *cidade*: nome da cidade – string
- *marca*: marca do veículo – string
- *modelo:* modelo do veículo – string

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

- *removido*: indica se o registro está logicamente removido. Pode assumir os valores ‘1’, para indicar que o registro está marcado como logicamente removido, ou ‘0’, para indicar que o registro não está marcado como removido. – tamanho: *string* de 1 byte.

- *prox:* armazena o RRN do próximo registro logicamente removido – tamanho: inteiro de 4 bytes
- *tamCidade:* armazena o tamanho da string de tamanho variável referente ao nome da cidade – tamanho: inteiro de 4 bytes
- *tamMarca:* armazena o tamanho da string de tamanho variável referente à marca do veículo – tamanho: inteiro de 4 bytes
- *tamModelo:* armazena o tamanho da string de tamanho variável referente ao modelo do veículo – tamanho: inteiro de 4 bytes.

Os dados são fornecidos juntamente com a especificação deste trabalho prático por meio de um arquivo .csv, sendo que sua especificação está disponível na página da disciplina. No arquivo .csv, o separador de campos é vírgula (,) e o primeiro registro especifica o que cada coluna representa (ou seja, contém a descrição do conteúdo das colunas).

Representação Gráfica dos Registros de Dados. O tamanho dos registros de dados deve ser de 97 bytes. Cada registro de dados deve ser representado da seguinte forma:

1 byte	4 bytes	4 bytes	4 bytes	4 bytes	2 bytes	4 bytes	1 byte	variável	4 bytes	1 byte	variável	4 bytes	1 byte	variável
removido	*prox*	*id*	*ano*	*qtt*	*sigla*	*tam Cidade*	*codC5 0*	*cidade*	*tam Marca*	*codC6 1*	*marca*	*tam Modelo*	*codC7 2*	*modelo*
0	1 ...	5 ...	9 ...	13 ...	17 18	...	...	...	...	...	...	...	...	...

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- As *strings* de tamanho variável são identificadas pelo seu tamanho e, portanto, não devem ser finalizadas com '\0'.
- O campo *id* não pode assumir valores nulos. Os demais campos podem assumir valores nulos. O arquivo .csv com os dados de entrada já garante essas características.
- Os valores nulos nos campos de tamanho fixo devem ser manipulados da seguinte forma. Os valores nulos devem ser representados pelo valor -1 quando

forem inteiros ou devem ser totalmente preenchidos pelo lixo '$' quando forem do tipo *string*.

- Os valores nulos nos campos de tamanho variável devem ser manipulados da seguinte forma: nada deve ser armazenado. Ou seja, caso um campo de tamanho variável tenha valor nulo, não deve ser armazenado o indicador de tamanho, o código que identifica o campo e o valor do campo. Por exemplo, caso o nome da cidade seja nulo, não devem ser armazenados no registro de dados os seguintes campos:

4 bytes	1 byte	variável
tam Cidade	*codC5 0*	*cidade*
...	...	...

- 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.

Arquivo Tipo 2
Descrição do Arquivo de Dados para Registros de Tamanho Variável

---

Um arquivo de dados possui um registro de cabeçalho e 0 ou mais registros de dados. A descrição do registro de cabeçalho do arquivo Tipo 2 é feita conforme a definição a seguir.

Registro de Cabeçalho. O registro de cabeçalho deve conter os seguintes campos:

- *status*: indica a consistência do arquivo de dados, devido à queda de energia, travamento do programa, etc. Pode assumir os valores '0', para indicar que o arquivo de dados está inconsistente, ou '1', para indicar que o arquivo de dados

está consistente. Ao 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.

- *topo:* armazena o *byte offset* de um registro logicamente removido, ou -1 caso não haja registros logicamente removidos – tamanho: inteiro de 8 bytes.
- *descricao:* indica que será feita a descrição dos metadados do arquivo de dados, juntamente com sua descrição simplificada. Deve armazenar o valor 'LISTAGEM DA FROTA DOS VEICULOS NO BRASIL' – tamanho: *string* de 40 bytes
- desC1: especifica a descrição detalhada do primeiro campo. Deve armazenar o valor 'CODIGO IDENTIFICADOR: ' – tamanho: *string* de 15 bytes
- desC2: especifica a descrição detalhada do segundo campo. Deve armazenar o valor 'ANO DE FABRICACAO: ' – tamanho: *string* de 19 bytes
- desC3: especifica a descrição detalhada do terceiro campo. Deve armazenar o valor 'QUANTIDADE DE VEICULOS: ' – tamanho: *string* de 24 bytes
- desC4: especifica a descrição detalhada do quarto campo. Deve armazenar o valor 'ESTADO: ' – tamanho: *string* de 8 bytes
- codC5: especifica a descrição simplificada do quinto campo. Deve armazenar o valor '0' – tamanho: *string* de 1 byte
- desC5: especifica a descrição detalhada do quinto campo. Deve armazenar o valor 'NOME DA CIDADE: ' – tamanho: *string* de 17 bytes
- codC6: especifica a descrição simplificada do sexto campo. Deve armazenar o valor '1' – tamanho: *string* de 1 byte
- desC6: especifica a descrição detalhada do sexto campo. Deve armazenar o valor 'MARCA DO VEICULO: ' – tamanho: *string* de 18 bytes
- codC7: especifica a descrição simplificada do sétimo campo. Deve armazenar o valor '2' – tamanho: *string* de 1 byte
- desC7: especifica a descrição detalhada do sétimo campo. Deve armazenar o valor 'MODELO DO VEICULO: ' – tamanho: *string* de 19 bytes
- *proxByteOffset*: armazena o valor do próximo *byte offset* disponível. Deve ser iniciado com o valor '0' e deve ser incrementado sempre que necessário – tamanho: inteiro de 8 bytes.

- *nroRegRem*: armazena o número de registros logicamente marcados como removidos. Deve ser iniciado com o valor '0' e ser incrementado quando necessário – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 190 bytes, representado da seguinte forma:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
status	topo								L	I	S	T	A	G	E	M		D	A
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
	F	R	O	T	A		D	O	S		V	E	I	C	U	L	O	S	
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
N	O		B	R	A	S	I	L	C	O	D	I	G	O		I	D	E	N
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
T	I	F	I	C	A	D	O	R	:		A	N	O		D	E		F	A
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
B	R	I	C	A	C	A	O	:		Q	U	A	N	T	I	D	A	D	E
100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
	D	E		V	E	I	C	U	L	O	S	:		E	S	T	A	D	O
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
:		0	N	O	M	E		D	A		C	I	D	A	D	E	:		1
140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
M	A	R	C	A		D	O		V	E	I	C	U	L	O	:		2	M
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
O	D	E	L	O		D	O		V	E	I	C	U	L	O	:			
180	181	182	183	184	185	186	187	188	189										
proxByteOffset						nroRegRem													

Observações Importantes.

- O registro de cabeçalho deve seguir estritamente a ordem definida na sua representação gráfica.
- Os campos são de tamanho fixo. Portanto, os valores que forem armazenados não devem ser finalizados por '\0'.
- 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 os campos de tamanho variável, deve ser usado o método indicador de tamanho.

Os campos de tamanho fixo são definidos da seguinte forma:

- *id*: código identificador – inteiro – tamanho: 4 bytes.

- *ano*: ano de fabricação do veículo – inteiro – tamanho: 4 bytes.
- *qtt*: quantidade de veículos – inteiro – tamanho: 4 bytes.
- *sigla*: sigla do estado no qual os veículos estão cadastrados – string – tamanho: 2 bytes.

Os campos de tamanho variável são definidos da seguinte forma:

- *cidade*: nome da cidade – string
- *marca*: marca do veículo – string
- *modelo:* modelo do veículo – string

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

- *removido*: indica se o registro está logicamente removido. Pode assumir os valores '1', para indicar que o registro está marcado como logicamente removido, ou '0', para indicar que o registro não está marcado como removido. – tamanho: *string* de 1 byte.
- *tamanhoRegistro:* número de bytes do registro – inteiro – tamanho: 4 bytes.
- *prox:* armazena o *byte offset* do próximo registro logicamente removido – tamanho: inteiro de 8 bytes.
- *tamCidade:* armazena o tamanho da string de tamanho variável referente ao nome da cidade – tamanho: inteiro de 4 bytes
- *tamMarca:* armazena o tamanho da string de tamanho variável referente à marca do veículo – tamanho: inteiro de 4 bytes
- *tamModelo:* armazena o tamanho da string de tamanho variável referente ao modelo do veículo – tamanho: inteiro de 4 bytes

Os dados são fornecidos juntamente com a especificação deste trabalho prático por meio de um arquivo .csv, sendo que sua especificação está disponível na página da disciplina. No arquivo .csv, o separador de campos é vírgula (,) e o primeiro registro especifica o que cada coluna representa (ou seja, contém a descrição do conteúdo das colunas).

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

1 byte	4 bytes	8 bytes	4 bytes	4 bytes	4 bytes	2 bytes	4 bytes	1 byte	variável	4 bytes	1 byte	variável	4 bytes	1 byte	variável
removido	*tamanho Registro*	*prox*	*id*	*ano*	*qtt*	*sigla*	*tam Cidade*	*codC5 0*	*cidade*	*tam Marca*	*codC6 1*	*marca*	*tam Modelo*	*codC7 2*	*modelo*
0	1 …	5 ...	13 …	17 …	21 …	25 26	…	…	…	…	…	…	…	…	…

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- As *strings* de tamanho variável são identificadas pelo seu tamanho e, portanto, não devem ser finalizadas com '\0'.
- O campo *id* não pode assumir valores nulos. Os demais campos podem assumir valores nulos. O arquivo .csv com os dados de entrada já garante essas características.
- Os valores nulos nos campos de tamanho fixo devem ser manipulados da seguinte forma. Os valores nulos devem ser representados pelo valor -1 quando forem inteiros ou devem ser totalmente preenchidos pelo lixo '$' quando forem do tipo *string*.
- Os valores nulos nos campos de tamanho variável devem ser manipulados da seguinte forma: nada deve ser armazenado. Ou seja, caso um campo de tamanho variável tenha valor nulo, não deve ser armazenado o indicador de tamanho, o código que identifica o campo e o valor do campo. Por exemplo, caso o nome da cidade seja nulo, não devem ser armazenados no registro de dados os seguintes campos:

4 bytes	1 byte	variável
tam Cidade	*codC5 0*	*cidade*
…	…	…

- 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 nesses arquivos binários. São definidas duas formas de organização diferentes para os arquivos binários, de forma que os alunos possam comparar as organizações em termos de implementação, utilização de espaço de armazenamento e recuperação dos dados.

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

Modularização. É importante modularizar o código. Trechos de programa que aparecerem várias vezes devem ser modularizados em funções e procedimentos.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

Na linguagem SQL, o comando CREATE TABLE é usado para criar uma tabela, a qual é implementada como um arquivo. Geralmente, uma tabela possui um nome (que corresponde ao nome do arquivo) e várias colunas, as quais correspondem aos campos dos registros do arquivo de dados. A funcionalidade [1] representa um exemplo de implementação do comando CREATE TABLE.

[1] Permita a leitura de vários registros obtidos a partir de um arquivo de entrada no formato csv e a gravação desses registros em um arquivo de dados de saída de acordo com o tipo desse arquivo. Se o arquivo for do *tipo1*, seus registros devem ser de tamanho fixo. Se o arquivo for do *tipo2*, seus registros devem ser de tamanho variável. O arquivo de entrada no formato csv é fornecido juntamente com a especificação do projeto, enquanto o arquivo de dados de saída deve ser gerado de acordo com as especificações deste trabalho prático. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo binário.

Entrada do programa para a funcionalidade [1]:

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

onde:

- `tipoArquivo`: tipo do arquivo, podendo assumir os valores `tipo1` (registros de tamanho fixo) ou `tipo2` (registros de tamanho variável), de acordo com a especificação deste trabalho prático.
- `arquivoEntrada.csv` é um arquivo .csv que contém os valores dos campos dos registros a serem armazenados no arquivo binário.
- `arquivoSaida.bin` é o arquivo binário gerado conforme as especificações descritas neste trabalho prático, podendo ser do `tipo1` ou do `tipo2`.

Saída caso o programa seja executado com sucesso:

```
Listar o arquivo de saída 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 tipo1 frota.csv frotaTipo1.bin
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo frotaTipo1.bin
```

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

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

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

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

[2] Permita a recuperação dos dados de todos os registros armazenados em um arquivo de dados de entrada de um determinado tipo, mostrando os dados de forma organizada na saída padrão para permitir a distinção dos campos e registros. O tratamento de 'lixo' deve ser feito de forma a permitir a exibição apropriada dos dados. Registros marcados como logicamente removidos não devem ser exibidos. O arquivo de dados de entrada deve ser percorrido apropriadamente de acordo com o seu tipo.

Entrada do programa para a funcionalidade [2]:

`2 tipoArquivo arquivoEntrada.bin`

onde:

- `tipoArquivo`: tipo do arquivo, podendo assumir os valores `tipo1` (registros de tamanho fixo) ou `tipo2` (registros de tamanho variável), de acordo com a especificação deste trabalho prático.
- `arquivoEntrada.bin` é o arquivo binário de um determinado tipo, o qual foi gerado conforme as especificações descritas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

Cada campo de cada registro deve ser mostrado em uma única linha da seguinte forma: descrição detalhada do campo: valor do campo. Caso o campo seja nulo, deve ser exibido: NAO PREENCHIDO. Deixe uma linha em branco depois que mostrar todos os campos de um registro. A ordem de exibição dos campos bem como os campos que devem ser exibidos podem ser vistos no **exemplo de execução.**

Mensagem de saída caso não existam registros:

`Registro inexistente.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução (é mostrado um exemplo ilustrativo)**:**

```
./programaTrab
2 tipo1 frotaTipo1.bin
MARCA DO VEICULO: VW
MODELO DO VEICULO: GOL 1.0
ANO DE FABRICACAO: 2006
NOME DA CIDADE: SANTA IZABEL DO OESTE
QUANTIDADE DE VEICULOS: 14
**** linha em branco ****
MARCA DO VEICULO: FIAT
MODELO DO VEICULO: SIENA 1.0
ANO DE FABRICACAO: 2021
NOME DA CIDADE: BELO HORIZONTE
QUANTIDADE DE VEICULOS: NAO PREENCHIDO
**** linha em branco ****
```

Conforme visto na funcionalidade [2], na linguagem SQL o comando SELECT é usado para listar os dados de uma tabela. Existem várias cláusulas que compõem o comando SELECT. Além das cláusulas SELECT e FROM, outra cláusula muito comum é a cláusula WHERE, que permite que seja definido um critério de busca sobre um ou mais campos, o qual é nomeado como critério de seleção.

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

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

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

A funcionalidade [3] representa um exemplo de implementação do comando SELECT considerando a cláusula WHERE. Como não existe índice definido sobre os campos dos registros, a implementação dessa funcionalidade consiste em percorrer sequencialmente o arquivo.

[3] Permita a recuperação dos dados de todos os registros de um arquivo de dados de entrada de um determinado tipo, de forma que esses registros satisfaçam um critério de busca determinado pelo usuário. Qualquer campo pode ser utilizado como forma de busca. Adicionalmente, a busca deve ser feita considerando um ou mais campos. Por exemplo, é possível realizar a busca considerando somente o campo *id* ou considerando os campos *cidade* e *modelo*. Esta funcionalidade pode retornar 0 registros (quando nenhum satisfaz ao critério de busca), 1 registro (quando apenas um satisfaz ao critério de busca), ou vários registros. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas ("). Para a manipulação de *strings* com aspas duplas, pode-se usar a função scan_quote_string disponibilizada na página do projeto da disciplina. Registros marcados como logicamente removidos não devem ser exibidos. O arquivo de dados de entrada deve ser percorrido apropriadamente de acordo com o seu tipo.

Sintaxe do comando para a funcionalidade [3]:

`3 tipoArquivo arquivoEntrada.bin n`
`nomeCampo`$_1$ `valorCampo`$_1$
`nomeCampo`$_2$ `valorCampo`$_2$
`...`
`nomeCampo`$_n$ `valorCampo`$_n$

onde:

- `tipoArquivo`: tipo do arquivo, podendo assumir os valores `tipo1` (registros de tamanho fixo) ou `tipo2` (registros de tamanho variável), de acordo com a especificação deste trabalho prático.
- `arquivoEntrada.bin` é o arquivo binário de um determinado tipo, o qual foi gerado conforme as especificações descritas neste trabalho prático.
- `n` é a quantidade de vezes que `nome do Campo` e `valor do Campo` podem repetir. Cada um dos `n` (`nomeCampo valorCampo`) deve ser especificado em uma linha diferente. Deve ser deixado um espaço em branco entre `nomeCampo` e `valorCampo`. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas (").

Saída caso o programa seja executado com sucesso:

`Cada campo de cada registro deve ser mostrado em uma única linha da seguinte forma: descrição detalhada do campo: valor do campo. Caso o campo seja nulo, deve ser exibido: NAO PREENCHIDO. Deixe uma linha em branco depois que mostrar todos os campos de um registro. A ordem de exibição dos campos bem como os campos que devem ser exibidos podem ser vistos no` **`exemplo de execução.`**

Mensagem de saída caso não seja encontrado o registro que contém o valor do campo ou o campo pertence a um registro que esteja removido:

`Registro inexistente.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
3 tipo1 frotaTipo1.bin 1
cidade "SANTA IZABEL DO OESTE"
MARCA DO VEICULO: VW
MODELO DO VEICULO: GOL 1.0
ANO DE FABRICACAO: 2006
NOME DA CIDADE: SANTA IZABEL DO OESTE
QUANTIDADE DE VEICULOS: 14
**** linha em branco ****
```

[4] Permita a recuperação dos dados de um registro de um arquivo de dados de entrada que seja do tipo1, a partir da identificação do RRN (número relativo do registro) do registro desejado pelo usuário. Por exemplo, o usuário pode solicitar a recuperação dos dados do registro de RRN = 2 ou do registro de RRN = 4. Essa funcionalidade pode retornar, no máximo, 1 registro. Os dados solicitados devem ser mostrados no mesmo formato definido para a funcionalidade [2]. Registros marcados como logicamente removidos não devem ser exibidos. Note que essa funcionalidade pode ser aplicada apenas a um arquivo de dados que possui registros de tamanho fixo. Isso porque registros de tamanho fixo podem ser identificados pelo RRN. Por outro lado, registros de tamanho variável não podem ser identificados pelo RRN.

Sintaxe do comando para a funcionalidade [4]:

`4 tipoArquivo arquivoEntrada.bin RRN`

onde:

- `tipoArquivo`: tipo do arquivo, podendo assumir somente o valor `tipo1` (registros de tamanho fixo), de acordo com a especificação deste trabalho prático.
- `arquivoEntrada.bin` é o arquivo binário do tipo1, o qual foi gerado conforme as especificações descritas neste trabalho prático.
- `RRN` é o número relativo do registro a ser recuperado.

Saída caso o programa seja executado com sucesso:

Cada campo de cada registro deve ser mostrado em uma única linha da seguinte forma: descrição detalhada do campo: valor do campo. Caso o campo seja nulo, deve ser exibido: NAO PREENCHIDO. Deixe uma linha em branco depois que mostrar todos os campos de um registro. A ordem de exibição dos campos bem como os campos que devem ser exibidos podem ser vistos no **exemplo de execução.**

Mensagem de saída caso não seja encontrado o registro ou o registro esteja removido:

Registro inexistente.

Mensagem de saída caso algum erro seja encontrado:

Falha no processamento do arquivo.

Exemplo de execução:

```
./programaTrab
4 tipo1 frotaTipo1.bin 0
cidade "SANTA IZABEL DO OESTE"
MARCA DO VEICULO: VW
MODELO DO VEICULO: GOL 1.0
ANO DE FABRICACAO: 2006
NOME DA CIDADE: SANTA IZABEL DO OESTE
QUANTIDADE DE VEICULOS: 14
**** 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: **ZWRM**
- **Turma 2** (terça-feira): código de matrícula: **C4QL**

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.

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).

Bom Trabalho!