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

Alunos PAE
Henrique Gomes Zanin
henrique.zanin@usp.br ou @hgzanin
João Paulo Clarindo dos Santos
jpcsantos@usp.br

Monitores
Guilherme Ramos Costa Paixão
guircp@usp.br ou telegram: @gp2112
Gustavo Lelli Guirao
gustavo.lelli@usp.br ou telegram: @gustavo_lelli

Trabalho Introdutório

Este trabalho tem como objetivo obter dados de um arquivo de entrada e gerar um arquivo binário com esses dados, bem como realizar operações de busca. Ele é um trabalho introdutório, de forma que será usado como base para o desenvolvimento de todos os demais trabalhos da disciplina.

O trabalho deve ser feito por 2 alunos da mesma turma. A solução deve ser proposta exclusivamente pelos alunos com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.

Fundamentos da disciplina de Bases de Dados

A disciplina de Organização de Arquivos é uma disciplina fundamental para a disciplina de Bases de Dados. A definição dos trabalhos práticos é feita considerando esse aspecto, ou seja, os trabalhos são especificados em termos de várias funcionalidades, e essas funcionalidades são relacionadas tanto com desafios enfrentados no mercado de trabalho quanto 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 podem 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 a **ataques cibernéticos**. Os dados originais podem ser obtidos a partir da URL https://www.kaggle.com/datasets/atharvasoundankar/global-cybersecurity-threats-2015-2024.

Descrição do Arquivo de Dados

Um arquivo de dados possui um registro de cabeçalho e 0 ou mais registros de dados. A descrição do registro de cabeçalho 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.
- *proxByteOffset*: armazena o valor do próximo *byte offset* disponível. Deve ser iniciado com o valor '0' e deve ser alterado sempre que necessário – tamanho: inteiro de 8 bytes.
- *nroRegArq*: armazena o número de registros não removidos presentes no arquivo. Ou seja, esse valor não inclui os registros logicamente marcados como removidos. Deve ser iniciado com o valor '0' e deve ser incrementado e decrementado 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 e decrementado quando necessário – tamanho: inteiro de 4 bytes.

- *descreveIdentificador*: descrição completa do campo *idAttack*. Identificador único que descreve cada ataque. Assume o valor "IDENTIFICADOR DO ATAQUE" – *string* de tamanho fixo: 23 bytes.
- *descreveYear*: descrição completa do campo *year*. Assume o valor "ANO EM QUE O ATAQUE OCORREU" – *string* de tamanho fixo: 27 bytes.
- *descreveFinancialLoss*: descrição completa do campo *financialLoss*. Assume o valor "PREJUIZO CAUSADO PELO ATAQUE" – *string* de tamanho fixo: 28 bytes.
- *codDescreveContry*: código da *keyword* que representa o campo *country*. Assume o valor "1" – *string* de tamanho fixo: 1 byte.
- *descreveCountry*: descrição completa do campo *country*. Assume o valor "PAIS ONDE OCORREU O ATAQUE" – *string* de tamanho fixo: 26 bytes.
- *codDescreveType*: código da *keyword* que representa o campo *attackType*. Assume o valor "2" – *string* de tamanho fixo: 1 byte.
- *descreveType*: descrição completa do campo *attackType*. Assume o valor "TIPO DE AMEACA A SEGURANCA CIBERNETICA" –*string* de tamanho fixo: 38 bytes.
- *codDescreveTargetIndustry*: código da *keyword* que representa o campo *targetIndustry*. Assume o valor "3" – *string* de tamanho fixo: 1 byte.
- *descreveTargetIndustry*: descrição completa do campo *targetIndustry*. Assume o valor "SETOR DA INDUSTRIA QUE SOFREU O ATAQUE" –*string* de tamanho fixo: 38 bytes.
- *codDescreveDefense*: código da *keyword* que representa o campo *defenseMechanism*. Assume o valor "4" – *string* de tamanho fixo: 1 byte.
- *descreveDefense*: descrição completa do campo *defenseMechanism*. Assume o valor "ESTRATEGIA DE DEFESA CIBERNETICA EMPREGADA PARA RESOLVER O PROBLEMA" –*string* de tamanho fixo: 67 bytes.

Explicação adicional sobre os campos que começam com o nome *descreve*.

Quando se define um arquivo de dados, é importante especificar também a semântica dos campos que compõem esse arquivo. Isso é feito pelo uso de *keywords* (*tags*). Uma *keyword* é especificada imediatamente antes do campo e define a semântica que explica o significado desse campo. Ela é usualmente utilizada em conjunto com qualquer método de organização de campos.

Dentre as vantagens do uso de *keywords*, tem-se: (i) o próprio campo fornece informação semântica a seu respeito; (ii) facilidade de identificar o conteúdo do arquivo; e (ii) facilidade de identificar valores nulos. Por outro lado, as *keywords* podem ocupar uma porção significativa do arquivo se forem definidas juntamente com os dados, causando desperdício de espaço de armazenamento. Neste sentido, as semânticas descritivas completas das *keywords* são definidas no registro de cabeçalho, conforme feita na especificação desta seção. No arquivo de dados, é utilizada apenas uma codificação que representa essas *keywords* de forma reduzida, diminuindo o desperdício de espaço de armazenamento.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 276 bytes, representado da seguinte forma:

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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
status	*topo*								*proxByteOffset*					
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
		nroRegArq				*nroRegRem*				I	D	E	N	T
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
I	F	I	C	A	D	O	R		D	O		A	T	A
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Q	U	E	A	N	O		E	M		Q	U	E		O
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
	A	T	A	Q	U	E		O	C	O	R	R	E	U
75	76	77	78	77	80	81	82	83	84	85	86	87	88	89
P	R	E	J	U	I	Z	O		C	A	U	S	A	D
90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
O		P	E	L	O		A	T	A	Q	U	E	1	P
105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
A	I	S		O	N	D	E		O	C	O	R	R	E
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134
U		O		A	T	A	Q	U	E	2	T	I	P	O
135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
	D	E		A	M	E	A	C	A		A		S	E
150	151	152	153	154	155	156	157	158	159	160	161	162	163	164
G	U	R	A	N	C	A		C	I	B	E	R	N	E
165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
T	I	C	A	3	S	E	T	O	R		D	A		I
180	181	182	183	184	185	186	187	188	189	190	191	192	193	194
N	D	U	S	T	R	I	A		Q	U	E		S	O
195	196	197	198	199	200	201	202	203	204	205	206	207	208	209
F	R	E	U		O		A	T	A	Q	U	E	4	E
210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
S	T	R	A	T	E	G	I	A		D	E		D	E
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	E	S	A		C	I	B	E	R	N	E	T	I	C
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254
A		E	M	P	R	E	G	A	D	A		P	A	R
255	256	257	258	259	260	261	262	263	264	265	266	267	268	269
A		R	E	S	O	L	V	E	R		O		P	R
270	271	272	273	274	275									
O	B	L	E	M	A									

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

Os campos de tamanho fixo são definidos da seguinte forma:

- *idAttack*: código identificador do ataque – inteiro – tamanho: 4 bytes.
- *year*: ano em que o ataque ocorreu – inteiro – tamanho: 4 bytes.
- *financialLoss*: prejuízo causado pelo ataque – float – tamanho: 4 bytes.

Os campos de tamanho variável são definidos da seguinte forma:

- *country*: país onde ocorreu o ataque – string
- *attackType:* tipo de ameaça à segurança cibernética – string
- *targetIndustry*: setor da indústria que sofreu o ataque – string
- *defenseMechanism*: estratégia de defesa cibernética empregada para resolver o problema – 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. Deve ser inicializado com o valor -1 quando necessário.

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

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

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
removido	*tamanhoRegistro*				*prox*								*idAttack*	
15	16	17	18	19	20	21	22	23	24	…	…	…	…	…
		year				*financialLoss*				*country* (variável)				
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
attackType (variável)					*targetIndustry* (variável)					*defenseMechanism* (variável)				

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- As *strings* de tamanho variável devem ser finalizadas com o delimitador.
- O campo *idAttack* não pode assumir valores nulos ou valores repetidos. 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 '$' quando forem do tipo *string*.
- Os valores dos campos de tamanho variável devem ser manipulados da seguinte forma. Se o valor não for nulo, o primeiro caractere deve ser o código da *keyword* que representa o campo. Por exemplo, para o campo *country*, tem-se: 1AUSTRALIA. Se o valor for nulo, não deve ser armazenado nada a respeito do campo, ou seja, não deve ser armazenado o código da *keyword* que representa o campo e o valor.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Sempre que houver lixo, ele deve ser identificado pelo caractere '$'. 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.

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. 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. A função binarioNaTela deve ser usada depois que o arquivo é fechado e antes de terminar a execução da funcionalidade.

Entrada do programa para a funcionalidade [1]:

`1 arquivoEntrada.csv arquivoSaida.bin`

onde:

- `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.

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 ataques.csv ataques.bin
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo ataques.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, mostrando os dados de forma organizada na saída padrão para permitir a distinção dos campos e registros. O tratamento de 'lixo' deve ser feito de forma a permitir a exibição apropriada dos dados. Registros marcados como logicamente removidos não devem ser exibidos.

Entrada do programa para a funcionalidade [2]:

`2 arquivoEntrada.bin`

onde:

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

Saída caso o programa seja executado com sucesso:

O valor de cada campo de cada registro deve ser mostrado em uma linha diferente, precedido pela descrição de seu conteúdo. Deve ser deixada uma linha em branco depois de cada registro. Caso o campo seja nulo, deve ser exibido o valor NADA CONSTA. Ver exemplo ilustrado no **exemplo de execução.**

Mensagem de saída caso não existam registros:

`Registro inexistente.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

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

```
./programaTrab
2 ataque.bin
IDENTIFICADOR DO ATAQUE: 1
ANO EM QUE O ATAQUE OCORREU: 2019
PAIS ONDE OCORREU O ATAQUE: CHINA
SETOR DA INDUSTRIA QUE SOFREU O ATAQUE: EDUCATION
TIPO DE AMEACA A SEGURANCA CIBERNETICA: PHISHING
PREJUIZO CAUSADO PELO ATAQUE: 80.53
ESTRATEGIA DE DEFESA CIBERNETICA EMPREGADA PARA RESOLVER O PROBLEMA: VPN
* deixar uma linha em branco *
IDENTIFICADOR DO ATAQUE: 2
ANO EM QUE O ATAQUE OCORREU: 2019
PAIS ONDE OCORREU O ATAQUE: CHINA
SETOR DA INDUSTRIA QUE SOFREU O ATAQUE: RETAIL
TIPO DE AMEACA A SEGURANCA CIBERNETICA: NADA CONSTA
PREJUIZO CAUSADO PELO ATAQUE: NADA CONSTA
ESTRATEGIA DE DEFESA CIBERNETICA EMPREGADA PARA RESOLVER O PROBLEMA: FIREWALL
* deixar uma 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 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 *idAttack* ou considerando os campos *year* e *attackType*. 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.

Sintaxe do comando para a funcionalidade [3]:

`3 arquivoEntrada.bin n`

`m1 nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{m1}$ `valorCampo`$_{m1}$

`m2 nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{m2}$ `valorCampo`$_{m2}$

`...`

`mn nomeCampo`$_1$ `valorCampo`$_1$ `...` `nomeCampo`$_{mn}$ `valorCampo`$_{mn}$

onde:

- `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 a busca deve acontecer.

- m é a quantidade de vezes que o par `nome do Campo` e `valor do Campo` pode repetir em uma busca. 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:

`O valor de cada campo de cada registro deve ser mostrado em uma linha diferente, precedido pela descrição de seu conteúdo. Deve ser deixada uma linha em branco depois de cada registro. Caso o campo seja nulo, deve ser exibido o valor NADA CONSTA. Adicionalmente, deve ser impressa uma linha preenchida com 10 valores "*" entre as buscas. Ver exemplo ilustrado no` **`exemplo de execução.`**

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

`Registro inexistente.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab
3 ataque.bin 2
1 idAttack 1
2 country "CHINA" targetIndustry "RETAIL"
IDENTIFICADOR DO ATAQUE: 1
ANO EM QUE O ATAQUE OCORREU: 2019
PAIS ONDE OCORREU O ATAQUE: CHINA
```

```
SETOR DA INDUSTRIA QUE SOFREU O ATAQUE: EDUCATION
TIPO DE AMEACA A SEGURANCA CIBERNETICA: PHISHING
PREJUIZO CAUSADO PELO ATAQUE: 80.53
ESTRATEGIA DE DEFESA CIBERNETICA EMPREGADA PARA RESOLVER O PROBLEMA: VPN
* deixar uma linha em branco *
**********
IDENTIFICADOR DO ATAQUE: 2
ANO EM QUE O ATAQUE OCORREU: 2019
PAIS ONDE OCORREU O ATAQUE: CHINA
SETOR DA INDUSTRIA QUE SOFREU O ATAQUE: RETAIL
TIPO DE AMEACA A SEGURANCA CIBERNETICA: NADA CONSTA
PREJUIZO CAUSADO PELO ATAQUE: NADA CONSTA
ESTRATEGIA DE DEFESA CIBERNETICA EMPREGADA PARA RESOLVER O PROBLEMA: FIREWALL
* deixar uma linha em branco *
IDENTIFICADOR DO ATAQUE: 2
ANO EM QUE O ATAQUE OCORREU: 2023
PAIS ONDE OCORREU O ATAQUE: CHINA
SETOR DA INDUSTRIA QUE SOFREU O ATAQUE: RETAIL
TIPO DE AMEACA A SEGURANCA CIBERNETICA: DDOS
PREJUIZO CAUSADO PELO ATAQUE: NADA CONSTA
ESTRATEGIA DE DEFESA CIBERNETICA EMPREGADA PARA RESOLVER O PROBLEMA:
ENCRYPTION
* deixar 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. O uso da webcam é obrigatório.

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

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

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

Instruções de entrega.

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

- página: https://runcodes.icmc.usp.br/
- **Turma 1** (segunda-feira): código de matrícula: **GFRN**
- **Turma 2** (terça-feira): código de matrícula: **7X6L**

O vídeo gravado deve ser submetido por meio da página da disciplina no e-disciplinas, no qual o grupo vai informar o nome de cada integrante, o número do grupo e um link que contém o vídeo gravado. Ao submeter o link, verifique se o mesmo pode ser acessado. Vídeos cujos links não puderem ser acessados receberão nota zero. Vídeos corrompidos ou que não puderem ser corretamente acessados receberão nota zero.

Critério de Correção

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

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

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

Restrições adicionais sobre o critério de correção.

- 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!