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
Eduardo Souza Rocha
eduardos.rocha17@usp.br *ou telegram: @edwolt*
Lucas de Medeiros Franca Romero
lucasromero@usp.br *ou telegram: @lucasromero*
Maria Júlia Soares De Grandi
maju.degrandi@usp.br ou *telegram: @majudegrandi*

Trabalho Introdutório

Este trabalho tem como objetivo obter dados de um arquivo de entrada e gerar um arquivo binário com esses dados. Ele é um trabalho introdutório, de forma que será usado como base para o desenvolvimento de todos os demais trabalhos da disciplina. Vislumbra-se iniciar os alunos da disciplina na manipulação dos arquivos .csv e .bin, bem como analisar o desempenho da dupla quanto ao desenvolvimento de um trabalho em equipe.

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 à segurança pública. Esses dados são mantidos pela Secretaria de Segurança Pública do Estado de São Paulo, e podem ser obtidos a partir da URL http://www.ssp.sp.gov.br/transparenciassp/Consulta.aspx.

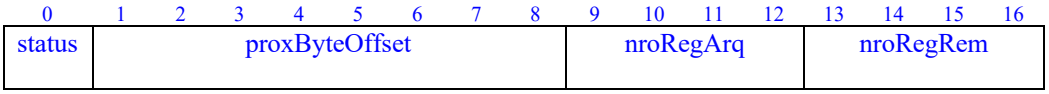
Descrição do Arquivo de Dados

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.
- *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 presentes no arquivo, incluindo registros logicamente marcados como removidos. Deve ser iniciado com o valor '0' e deve 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 quando necessário – tamanho: inteiro de 4 bytes.

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 17 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.

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 delimitador entre campos. O delimitador de registro é o caractere '#' e o delimitador de campo é o caractere '|'.

Os campos de tamanho fixo são definidos da seguinte forma:

- *idCrime*: código identificador do crime – inteiro – tamanho: 4 bytes.
- *dataCrime*: data da ocorrência do crime no formato DD/MM/AAAA – *string* – tamanho: 10 bytes.
- *numeroArtigo*: número do artigo no código penal que corresponda ao crime ocorrido – inteiro – tamanho: 4 bytes.
- *marcaCelular:* marca do celular que foi furtado/roubado, caso a ocorrência tenha relação com um desses crimes – string – tamanho: 12 bytes

Os campos de tamanho variável são definidos da seguinte forma:

- *lugarCrime*: lugar no qual o crime ocorreu – string
- *descricaoCrime:* descrição detalhada do crime – string

Adicionalmente, o seguinte campo de tamanho fixo também compõe cada registro. Esse campo é necessário 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. – *string* – tamanho: 1 byte

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

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

1 byte	4 bytes	10 bytes	4 bytes	12 bytes	variável	variável	1 byte
removido	idCrime	dataCrime	numeroArtigo	marcaCelular	lugarCrime	descricaoCrime	#
0	1 … 4	5 … 14	15 …18	19 ... 30	…	…	

Observações Importantes.

- Cada registro de dados deve seguir estritamente a ordem definida na sua representação gráfica.
- Cada registro de dados deve ser finalizado com o delimitador '#'
- As *strings* de tamanho variável devem ser finalizadas com o delimitador '|'. Portanto, não devem ser finalizadas com '\0'.
- Caso o valor do campo *marcaCelular* tenha tamanho menor do que 12 bytes, os bytes não utilizados devem ser preenchidos com o caractere '$'. Portanto, o valor não deve ser finalizado com '\0'.
- O campo *idCrime* não pode assumir valores nulos. Os demais campos podem assumir valores nulos. O arquivo .csv com os dados de entrada já garante essas características.
- O campo *idCrime* não pode assumir valores repetidos. Os demais campos podem assumir 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 caractere '$' quando forem do tipo *string*.
- Os valores nulos nos campos de tamanho variável devem ser manipulados da seguinte forma: deve ser armazenado apenas o delimitador '|'.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Sempre que houver lixo, ele deve ser identificado pelo caractere '$'. 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.

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.

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 crime.csv crime.bin
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo crime.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 única linha, e deve ser separado por vírgula. Caso o campo seja nulo, deve ser exibido: NULO. Ordem de exibição dos campos: idCrime, dataCrime, numeroArtigo, lugarCrime, descriçãoCrime, marcaCelular, conforme 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 crime.bin
1, 08/04/2017, 157, SAO CARLOS, ROUBO, NOKIA
2, 14/08/2022, 171, BELO HORIZONTE, ESTELIONATO CONTRA IDOSO, NULO
...
```

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.

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://run.codes/Users/login
- **Turma 1** (segunda-feira): código de matrícula: **NJMU**
- **Turma 2** (terça-feira): código de matrícula: **9NMB**

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!