Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Algoritmos e Estruturas de Dados II
docente
Profa. Dra. Cristina Dutra de Aguiar Ciferri (cdac@icmc.usp.br)
aluno PAE
Victor Hugo Andrade Soares (victorhugoasoares@usp.br)
monitor
João Vitor dos Santos Tristão (joaovitortristao@usp.br)

Terceiro Trabalho Prático

Este trabalho tem como objetivo ordenar um arquivo de dados e implementar as operações cosequenciais de *merging* e *matching*.

*O trabalho deve ser feito **individualmente**. A solução deve ser proposta exclusivamente pelo aluno com base nos conhecimentos adquiridos nas aulas. Consulte as notas de aula e o livro texto quando necessário.*

Programa

Descrição Geral. Implemente um programa em C que ofereça uma interface por meio da qual o usuário possa realizar a ordenação de um arquivo de dados de entrada, bem como implementar as operações cosequencias de *merging* e *matching*. Deve-se levar em consideração a descrição e a organização do arquivo de dados especificados no primeiro trabalho prático.

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 "programaTrab3". Essas orientações devem ser seguidas uma vez que a correção do funcionamento do programa se dará de forma automática.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

[8] Realize a ordenação interna dos dados de um arquivo de dados, considerando os valores do campo *nroInscricao*. A ordenação deve ser numérica, desde que o campo *nroInscricao* é do tipo inteiro (4 bytes). Deve ser considerado que o arquivo completo cabe em RAM. Como entrada dessa funcionalidade, tem-se um arquivo de dados de entrada, o qual foi gerado segundo as especificações definidas no primeiro trabalho prático. Como saída dessa funcionalidade, tem-se um novo arquivo de dados de saída, o qual possui os mesmos campos que o arquivo de entrada, porém cujos valores dos registros são ordenados de forma crescente com base no campo *nroInscricao*. Adicionalmente, o arquivo de saída não deve conter os registros logicamente removidos. A ordenação interna pode ser realizada usando-se qualquer algoritmo de ordenação interna eficiente existente na literatura (ex.: *quicksort, heapsort*). A implementação dessa funcionalidade deve ser feita de forma eficiente. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela1 ou binarioNaTela2, ambas disponibilizadas na página do projeto da disciplina, para mostrar a saída do arquivo binário de saída.

Entrada do programa para a funcionalidade [8]:
`8 arquivoEntrada.bin arquivoSaida.bin`
onde:
- `arquivoEntrada.bin` é um arquivo binário de entrada que segue as mesmas especificações do primeiro trabalho prático, e que contém dados desordenados e registros logicamente removidos.
- `arquivoSaida.bin` é um arquivo binário de saída que segue as mesmas especificações definidas no primeiro trabalho prático, e que contém dados ordenados de forma crescente nos valores do campo *nroInscricao* e não contém registros logicamente removidos.

Saída caso o programa seja executado com sucesso:
`Listar o arquivo binário arquivoSaida.bin.`
Mensagem de saída caso algum erro seja encontrado:
`Falha no processamento do arquivo.`
Exemplo de execução:
```
./programaTrab3
8 arquivo1.bin arquivo2.bin
usar a função binarioNaTela1 ou binarioNaTela2 antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivo2.bin, o qual foi ordenado de forma crescente nos valores do campo idServidor.
```

[9] Realize a operação cosequencial de merging (união) de dois arquivos de dados, considerando os valores do campo *nroInscricao*. Como entrada dessa funcionalidade, tem-se dois arquivos de dados de entrada, `arquivoEntrada1` e `arquivoEntrada2`, os quais devem ser gerados segundo as especificações definidas na funcionalidade [8]. Como saída dessa operação, tem-se um arquivo de dados de saída totalmente ordenado, `arquivoSaida`, o qual possui os mesmos campos que cada um dos arquivos de dados de entrada, porém armazena todos os registros contidos em *arquivo1* ou em *arquivo2* ou em ambos *arquivo1* e *arquivo2*. Esses registros encontram-se ordenados de forma crescente considerando os valores do campo *nroInscricao*. A ordenação deve ser numérica, desde que o campo *nroInscricao* é do tipo inteiro (4 bytes). Em casos de empate para os valores de *nroInscricao*, deve ser armazenado em `arquivoSaida` somente um registro: o registro de `arquivoEntrada1`. A implementação dessa funcionalidade deve ser feita de forma eficiente. Ou seja, segundo a matéria estudada na disciplina, a operação cosequencial de *merging* deve incorporar os seguintes pontos importantes: inicialização, sincronização e gerenciamento de condição de fim de arquivo. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela1 ou binarioNaTela2, ambas disponibilizadas na página do projeto da disciplina, para mostrar a saída do arquivo binário de saída.

Entrada do programa para a funcionalidade [9]:

`9 arquivoEntrada1.bin arquivoEntrada2.bin arquivoSaida.bin`

onde:

- `arquivoEntrada1.bin e arquivoEntrada2.bin` são arquivos binários de entrada que seguem as mesmas especificações do primeiro trabalho prático, e que foram ordenados de acordo com as especificações da funcionalidade [8].
- `arquivoSaida.bin` é um arquivo binário de saída que segue as mesmas especificações definidas no primeiro trabalho prático, e que contém todos os registros de `arquivoEntrada1` ou de `arquivoEntrada2` ou de ambos `arquivoEntrada1` e `arquivoEntrada2`. Os registros de `arquivoSaida`.bin encontram-se ordenados de forma crescente nos valores do campo *nroInscricao*. Não devem existir registros repetidos no arquivo de saída.

Saída caso o programa seja executado com sucesso:

`Listar o arquivo binário arquivoSaida.bin.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab3
9 arquivoEntrada1.bin arquivoEntrada2.bin arquivoSaida.bin
usar a função binarioNaTela1 ou binarioNaTela2 antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoSaida.bin.
```

[10] Realize a operação cosequencial de *matching* (interesecção) de dois arquivos de dados, considerando os valores do campo *nroInscricao*. Como entrada dessa funcionalidade, tem-se dois arquivos de dados de entrada, `arquivoEntrada1` e `arquivoEntrada2`, os quais devem ser gerados segundo as especificações definidas na funcionalidade [8]. Como saída dessa operação, tem-se um arquivo de dados de saída totalmente ordenado, `arquivoSaida`, o qual possui os mesmos campos que cada um dos arquivos de dados de entrada, porém armazena apenas todos os registros que estejam presentes em ambos *arquivo1* e *arquivo2*. Esses registros encontram-se ordenados de forma crescente considerando os valores do campo *nroInscricao*. A ordenação deve ser numérica, desde que o campo *nroInscricao* é do tipo inteiro (4 bytes). Em casos de empate para os valores de *idServidor*, deve ser armazenado em `arquivoSaida` somente um registro: o registro de `arquivoEntrada1`. A implementação dessa funcionalidade deve ser feita de forma eficiente. Ou seja, segundo a matéria estudada na disciplina, a operação cosequencial de *matching* deve incorporar os seguintes pontos importantes: inicialização, sincronização e gerenciamento de condição de fim de arquivo. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela1 ou binarioNaTela2, ambas disponibilizadas na página do projeto da disciplina, para mostrar a saída do arquivo binário de saída.

Entrada do programa para a funcionalidade [10]:

`10 arquivoEntrada1.bin arquivoEntrada2.bin arquivoSaida.bin`

onde:

- `arquivoEntrada1.bin e arquivoEntrada2.bin` são arquivos binários de entrada que seguem as mesmas especificações do primeiro trabalho prático, e que foram ordenados de acordo com as especificações da funcionalidade [8].
- `arquivoSaida.bin` é um arquivo binário de saída que segue as mesmas especificações definidas no primeiro trabalho prático, e que contém apenas todos os registros presentes em ambos `arquivoEntrada1` e `arquivoEntrada2`. Os registros de `arquivoSaida`.bin encontram-se ordenados de forma crescente nos valores do campo *nroInscricao*. Não devem existir registros repetidos no arquivo de saída.

Saída caso o programa seja executado com sucesso:

`Listar o arquivo binário arquivoSaida.bin.`

Mensagem de saída caso algum erro seja encontrado:

`Falha no processamento do arquivo.`

Exemplo de execução:

```
./programaTrab3
10 arquivoEntrada1.bin arquivoEntrada2.bin arquivoSaida.bin
usar a função binarioNaTela1 ou binarioNaTela2 antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoSaida.bin.
```

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 e lidos campo a campo. Ou seja, não é possível escrever e ler os dados registro a registro.

[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. 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 nas páginas 96 a 107 do 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. Assim, a forma mais simples de se fazer o arquivo makefile é:

```
all:
gcc -o programaTrab2 *.c
run:
./programaTrab2
```

Lembrando que *.c já engloba todos os arquivos .c presentes no seu zip."

Instruções de entrega. A entrega deve ser feita via [run.codes]:

- página: https://run.codes/Users/login
- código de matrícula: **91X6**

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.

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 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 !