Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Estrutura de Dados III
Profa. Dra. Cristina Dutra de Aguiar Ciferri
PAE: João Pedro de Carvalho Castro
Monitor: Matheus Carvalho Raimundo

Trabalho Prático

Este trabalho tem como objetivo implementar as operações cosequenciais de *merging*, *matching* e *multiway merging*, bem como realizar a ordenação de grandes arquivos de dados.

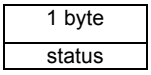
O trabalho deve ser feito em grupo de 4 alunos. A solução deve ser proposta exclusivamente pelo grupo com base nos conhecimentos adquiridos ao longo das aulas. Consulte as notas de aula e o livro texto quando necessário.

Descrição do arquivo de dados

Registro de cabeçalho. O registro de cabeçalho deve conter o seguinte campo:

- *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: 1 byte.

Representação gráfica do registro de cabeçalho. O tamanho do registro de cabeçalho deve ser de 1 byte, representado da seguinte forma:
Tamanho do registro de tamanho fixo: 1 byte.



Observação. O registro de cabeçalho deve seguir estritamente a sua especificação. Ou seja, o campo deve se chamar *status* e deve ter o tamanho de 1 byte.

Registros de dados. Deve ser considerada a organização descrita a seguir:

- Campos de tamanho fixo.
- Registros de tamanho fixo.

Domínio. Cada registro do arquivo de dados deve conter dados relacionados a um assunto escolhido pelo grupo. Ou seja, o domínio é livre. Os grupos devem escolher diferentes domínios de dados para fazerem os seus trabalhos.

Importante. O domínio escolhido pelo grupo deve ser previamente validado pelo aluno PAE da disciplina. Para tanto, o grupo deve enviar uma mensagem para João Pedro de Carvalho Castro (jp.carvalhocastro@outlook.com) com o assunto [Estruturas de Dados III] Domínio da Aplicação. No corpo da mensagem, o grupo deve explicar o domínio e o significado de cada campo. Os domínios de dados de cada grupo serão mantidos atualizados na página da disciplina.

Detalhamento dos registros. Cada registro do arquivo de dados deve conter a seguinte estruturação de campos:

- *campo1*: inteiro sem sinal (somente valores positivos) – tamanho: 4 bytes
- *campo2*: *string* de 30 caracteres – tamanho: 30 bytes
- *campo3*: string de 20 caracteres – tamanho: 20 bytes
- *campo4*: *string* de 10 caracteres no formato ##/##/#### (DD/MM/AAAA, ex.: 21/12/2012) – tamanho: 10 bytes

Representação gráfica dos registros de dados

Tamanho do registro de tamanho fixo: 64 bytes.

<table>
<tr><td colspan="2">4 bytes</td><td colspan="2">30 bytes</td><td colspan="2">20 bytes</td><td colspan="2">10 bytes</td></tr>
<tr><td colspan="2">campo1</td><td colspan="2">campo2</td><td colspan="2">campo3</td><td colspan="2">campo 4</td></tr>
<tr><td>0</td><td>1</td><td colspan="4">...</td><td>62</td><td>63</td></tr>
</table>

Observação. O registro de dados de entrada deve seguir estritamente a ordem definida na sua representação gráfica. Os nomes dos atributos também devem seguir estritamente os nomes definidos na especificação dos mesmos. O tamanho do registro de tamanho fixo é de 64 bytes, conforme ilustrado pelos byte offsets 0, 1, ..., 62, 63.

Programa

Descrição geral. Implemente um programa em C que ofereça uma interface, via linha de comando, por meio da qual o usuário possa realizar as operações de *merging*, *matching* e *multiway merging*, além da operação de *sort-merge* externo. Deve-se levar em consideração a descrição do arquivo de dados especificada anteriormente. A definição da sintaxe de cada comando deve seguir estritamente as especificações definidas em cada funcionalidade.

Importante. A definição da sintaxe de cada comando bem como sua saída na saída padrão devem seguir estritamente as especificações definidas em cada funcionalidade. Para especificar a sintaxe de execução, considere que o programa seja chamado de "programaTrab" (entretanto, o grupo pode definir qualquer outro nome para o programa). Essas orientações devem ser seguidas uma vez que a correção do funcionamento do programa se dará de forma automática por meio de *scripts* de execução. De forma geral, o primeiro parâmetro a ser passado para o programa por meio da linha de comando é sempre o identificador de suas funcionalidades, conforme especificado a seguir.

Descrição Específica. O programa deve oferecer as seguintes funcionalidades:

[1] Realize a geração automática de *n* registros do arquivo de dados de entrada, sem qualquer ordenação dos dados. O valor de *n* deve ser passado como parâmetro, e deve ser de, no mínimo, 6.000 registros. Como saída dessa operação, deve ser gerado um arquivo de dados binário contendo todos os registros.

Todos os campos devem ser preenchidos, ou seja, nenhum campo pode ter valores nulos. Todos os valores de campos devem ser escritos em letras maiúsculas, sem acento e sem o uso de caracteres especiais. Se o valor de algum campo não ocupar todo o espaço disponível para ele, o restante do espaço pode conter 'lixo'.

Campo1, *campo2*, *campo3* e *campo4* são campos de chave secundária, ou seja, esses campos podem assumir valores repetidos. Devem ser gerados 30% de valores repetidos para *campo1*, 25% de valores repetidos para *campo2*, 20% de valores repetidos para *campo3* e 15% de valores repetidos para *campo4*. Considerando especificamente *campo1*, o domínio de seus valores pode variar de 1 a 50.000.

Sintaxe do comando para a funcionalidade [1]:

```
./programaTrab 1 'nomeArquivoQueSeraGerado' n
```

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo gerado.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento.
```

Exemplos de execução:

```
./programaTrab 1 'arquivoEntrada1' 6000
Arquivo gerado.
./programaTrab 1 'arquivoEntrada2' 8000
Arquivo gerado.
```

[2] Permita a recuperação dos dados, de todos os registros, armazenados no arquivo de dados, 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.

Sintaxe do comando para a funcionalidade [2]:

```
./programaTrab 2 'nomeArquivoQueSeraExibido'
```

Saída caso o programa seja executado com sucesso:

```
Cada registro deve ser mostrado em uma linha e os seus campos devem ser mostrados de forma sequencial, separados por um espaço em branco.
```

Mensagem de saída caso não existam registros:

```
Arquivo vazio.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento.
```

Exemplos de execução:

```
./programaTrab 2 'arquivoEntrada1'
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa yyyyyyyyyyyyyyyyyyyy 21/12/2012
17 sssssssssssssssssssssssssssss aaaaaaaaaaaaaaaaaaa 18/08/2008
10 yyyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyyy 17/01/2009
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa ttttttttttttttttttt 28/12/2018
13 bbbbbbbbbbbbbbbbbbbbbbbbbbbbb ccccccccccccccccccc 14/07/2017
...

./programaTrab 2 'arquivoEntrada2'
13 wwwwwwwwwwwwwwwwwwwwwwwwwwwww wwwwwwwwwwwwwwwwwwww 14/04/2004
04 eeeeeeeeeeeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeee 09/01/2009
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa ddddddddddddddddddddd 03/05/2018
...
```

[3] Realize a ordenação interna dos dados de um arquivo de dados de entrada, considerando primeiramente os valores de *campo1*, depois os valores de *campo2*, depois os valores de *campo3* e depois os valores de *campo4*. Ou seja, a ordenação deve ser feita considerando os valores de *campo1*. Quando os valores de *campo1* forem iguais, então deve-se realizar a ordenação considerando os valores de *campo2*. Quando os valores de *campo1* e *campo2* forem iguais, deve-se realizar a ordenação considerando os valores de *campo3*. Quando os valores de *campo1*, *campo2* e *campo3* forem iguais, deve-se considerar os valores de *campo4*.

Como entrada dessa operação, tem-se um arquivo de dados de entrada, o qual foi gerado segundo as especificações definidas na funcionalidade [1]. Como saída dessa operação, tem-se um arquivo de dados de saída, o qual possui os mesmos campos que o arquivo de entrada, porém cujos valores dos registros são ordenados.

A ordenação interna pode ser realizada usando-se qualquer algoritmo de ordenação interna eficiente existente na literatura (ex.: *quicksort, heapsort*).

Sintaxe do comando para a funcionalidade [3]:

```
./programaTrab 3 'nomeArquivoEntrada''nomeArquivoQueSeraGerado'
```

Saída caso o programa seja executado com sucesso:

```
Arquivo de dados de entrada ordenado.
```

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo gerado.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento.
```

Exemplos de execução:

```
./programaTrab 3 'arquivoEntrada1' 'arquivoOrdenadoEntrada1'
Arquivo gerado.
./programaTrab 3 'arquivoEntrada2' 'arquivoOrdenadoEntrada2'
Arquivo gerado.
```

Exemplos de execução da funcionalidade [2], após a execução da funcionalidade [3]:

```
./programaTrab 2 'arquivoOrdenadoEntrada1'
10 yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyyy 17/01/2009
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa ttttttttttttttttttt 28/12/2018
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa yyyyyyyyyyyyyyyyyyyy 21/12/2012
13 bbbbbbbbbbbbbbbbbbbbbbbbbbbbb ccccccccccccccccccc 14/07/2017
17 sssssssssssssssssssssssssssss aaaaaaaaaaaaaaaaaaa 18/08/2008
...

./programaTrab 2 'arquivoOrdenadoEntrada2'
04 eeeeeeeeeeeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeee 09/01/2009
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa ddddddddddddddddddddd 03/05/2018
13 wwwwwwwwwwwwwwwwwwwwwwwwwwww wwwwwwwwwwwwwwwwwwww 14/04/2004
...
```

[4] Realize a operação cosequencial de *merging* (união) considerando primeiramente os valores de *campo1*, depois os valores de *campo2*, depois os valores de *campo3* e depois os valores de *campo4*. Ou seja, a operação de *merging* deve ser feita considerando os valores de *campo1*. Quando os valores de *campo1* forem iguais, então deve-se realizar a operação de *merging* considerando os valores de *campo2*. Quando os valores de *campo1* e *campo2* forem iguais, deve-se realizar a operação de *merging* considerando os valores de *campo3*. Quando os valores de *campo1*, *campo2* e *campo3* forem iguais, deve-se realizar a operação de *merging* considerando os valores de *campo4*.

Como entrada dessa operação, tem-se dois arquivos ordenados de dados, *arquivo1* e *arquivo2*, os quais devem ser gerados segundo as especificações definidas na funcionalidade [1] e ordenados segundo a funcionalidade [3]. Como saída dessa operação, tem-se um arquivo de dados de saída totalmente ordenado, 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*.

Lembrando que, 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. Adicionalmente, neste projeto, considere que valores repetidos não representam erros.

Sintaxe do comando para a funcionalidade [4]:

```
./programaTrab 4 'nomeArquivoEntrada1' 'nomeArquivoEntrada2'
'nomeArquivoQueSeraGerado'
```

Saída caso o programa seja executado com sucesso:

```
Deve ser gerado o arquivo de saída contendo o resultado da operação
cosequencial de merging.
```

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo gerado.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento.
```

Exemplos de execução:

```
./programaTrab 4 'arquivoOrdenadoEntrada1' 'arquivoOrdenadoEntrada2'
'arquivoMerge'
Arquivo gerado.
```

Exemplo de execução da funcionalidade [2], após a execução da funcionalidade [4]:

```
./programaTrab 2 'arquivoMerge'
04 eeeeeeeeeeeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeee 09/01/2009
10 yyyyyyyyyyyyyyyyyyyyyyyyyyyy yyyyyyyyyyyyyyyyyyy 17/01/2009
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaa ddddddddddddddddddd 03/05/2018
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaa ttttttttttttttttttt 28/12/2018
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaa yyyyyyyyyyyyyyyyyyy 21/12/2012
13 bbbbbbbbbbbbbbbbbbbbbbbbbbbb ccccccccccccccccccc 14/07/2017
13 wwwwwwwwwwwwwwwwwwwwwwwwwwww wwwwwwwwwwwwwwwwwww 14/04/2004
17 sssssssssssssssssssssssssssss aaaaaaaaaaaaaaaaaaa 18/08/2008
...
```

[5] Realize a operação cosequencial de *matching* (intersecção) considerando os valores de *campo1* apenas. Como entrada dessa operação, tem-se dois arquivos de dados, *arquivo1* e *arquivo2*, os quais devem ser gerados segundo as especificações definidas na funcionalidade [1] e ordenados segundo a funcionalidade [3]. Como saída dessa operação, tem-se um arquivo de dados de saída totalmente ordenado, o qual possui os mesmos campos que cada um dos arquivos de dados de entrada, porém armazena todos os registros contidos em *arquivo1* e em *arquivo2* segundo os valores de *campo1*.

Quando os valores de *campo1* forem iguais, os demais campos do arquivo de dados de saída devem ser ordenados considerando os valores de *campo2*, depois os valores de *campo3* e depois os valores de *campo4*. Ou seja, a operação de *matching* deve ser feita considerando os valores de *campo1*. Quando os valores de *campo1* forem iguais, então deve-se realizar a ordenação considerando os valores de *campo2*. Quando os valores de *campo1* e *campo2* forem iguais, deve-se realizar a ordenação considerando os valores de *campo3*. Quando os valores de *campo1*, *campo2* e *campo3* forem iguais, deve-se realizar a ordenação considerando os valores de *campo4*.

Lembrando que, 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. Adicionalmente, neste projeto, considere que valores repetidos não representam erros.

Sintaxe do comando para a funcionalidade [5]:

```
./programaTrab 5 'nomeArquivoEntrada1' 'nomeArquivoEntrada2'
'nomeArquivoQueSeraGerado'
```

Saída caso o programa seja executado com sucesso:

Deve ser gerado o arquivo de saída contendo o resultado da operação cosequencial de matching.

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo gerado.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento.
```

Exemplos de execução:

```
./programaTrab 5 'arquivoOrdenadoEntrada1' 'arquivoOrdenadoEntrada2'
'arquivoMatch'
Arquivo gerado.
```

Exemplo de execução da funcionalidade [2], após a execução da funcionalidade [5]:

```
./programaTrab 2 'arquivoMatch'
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa ddddddddddddddddddddd 03/05/2018
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa ttttttttttttttttttt 28/12/2018
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaa yyyyyyyyyyyyyyyyyyyy 21/12/2012
13 bbbbbbbbbbbbbbbbbbbbbbbbbbbbb ccccccccccccccccccc 14/07/2017
13 wwwwwwwwwwwwwwwwwwwwwwwwwwww wwwwwwwwwwwwwwwwwwww 14/04/2004
...
```

[6] Realize a operação *multiway merging* (união) de muitas listas, considerando primeiramente os valores de *campo1*, depois os valores de *campo2*, depois os valores de *campo3* e depois os valores de *campo4*. Ou seja, a operação de *multiway merging* deve ser feita considerando os valores de *campo1*. Quando os valores de *campo1* forem iguais, então deve-se realizar a operação de *multiway merging* considerando os valores de *campo2*. Quando os valores de *campo1* e *campo2* forem iguais, deve-se realizar a operação de *multiway merging* considerando os valores de *campo3*. Quando os valores de *campo1*, *campo2* e *campo3* forem iguais, deve-se realizar a operação de *multiway merging* considerando os valores de *campo4*..

Como entrada dessa operação, tem-se k arquivos ordenados de dados, *arquivo1*, *arquivo2*, ..., *arquivoK*, os quais devem ser gerados segundo as especificações definidas na funcionalidade [1] e ordenados segundo a funcionalidade [3]. Como saída dessa operação, tem-se um arquivo de dados de saída totalmente ordenado, 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 ... ou em *arquivoK* ou em ambos *arquivo1* e *arquivo2* e ... e *arquivok*.

Lembrando que, segundo a matéria estudada na disciplina, a operação *multiway merging* de muitas listas deve ser implementada como uma árvore de seleção. Adicionalmente, neste projeto, considere que valores repetidos não representam erros. Considere também que os k arquivos podem ser abertos e manipulados simultaneamente.

Sintaxe do comando para a funcionalidade [6]:

```
./programaTrab 6 'nomeArquivoEntrada1' 'nomeArquivoEntrada2' ...
'nomeArquivoEntradaK''nomeArquivoQueSeraGerado'
```

Saída caso o programa seja executado com sucesso:

Deve ser gerado o arquivo de saída contendo o resultado da operação de multiway merging de muitas listas.

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo gerado.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento.
```

Exemplo de execução:

```
./programaTrab 6 'arquivoOrdenadoEntrada1' 'arquivoOrdenadoEntrada2'
'arquivoOrdenadoEntrada3' 'arquivoOrdenadoEntrada4'
'arquivoMultiwayMerge'
Arquivo gerado.
```

[7] Realize a ordenação externa de um arquivo de dados de entrada, usando o algoritmo de *sort-merge* externo e considerando primeiramente os valores de *campo1*, depois os valores de *campo2*, depois os valores de *campo3* e depois os valores de *campo4*. Ou seja, a ordenação deve ser feita considerando os valores de *campo1*. Quando os valores de *campo1* forem iguais, então deve-se realizar fazer a ordenação considerando os valores de *campo2*. Quando os valores de *campo1* e *campo2* forem iguais, deve-se fazer a ordenação considerando os valores de *campo3*. Quando os valores de *campo1*, *campo2* e *campo3* forem iguais, deve-se fazer a ordenação considerando os valores de *campo4*.

Como entrada dessa operação, tem-se um arquivo de dados de entrada, o qual deve ser gerado segundo as especificações definidas na funcionalidade [1]. Como saída dessa operação, tem-se um arquivo de dados de saída, o qual possui os mesmos campos que o arquivo de entrada, porém cujos valores dos registros são ordenados.

Nessa funcionalidade, deve-se considerar páginas de disco de 16.000 bytes, e buffers de tamanho de 4 páginas de disco.

O arquivo de entrada, bem como todos os subarquivos gerados, devem permanecer armazenados em disco após a execução da funcionalidade. Esses subarquivos devem ser nomeados usando-se o prefixo "sub_".

Sintaxe do comando para a funcionalidade [7]:

```
./programaTrab 7 'nomeArquivoEntrada' 'nomeArquivoQueSeraGerado'
```

Saída caso o programa seja executado com sucesso:

Devem ser gerados todos os subarquivos usados na ordenação, bem como o arquivo de saída contendo o resultado da operação de ordenação externa segundo o algoritmo *sort-merge* externo.

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo gerado.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento.
```

Exemplo de execução:

```
./programaTrab 7 'arquivoEntrada1' 'arquivoSortMerge'
Arquivo gerado.
```

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 deve 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] Não deve haver a manipulação de valores nulos.

[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 escritos e lidos campo a campo. Pode-se usar também a serialização (memcpy).

[7] Os integrantes do grupo devem constar como comentário no início do código (i.e. NUSP e nome de cada integrante do grupo). 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 interface deve obrigatoriamente ser via linha de comando, sendo que a sintaxe de cada comando deve seguir estritamente as especificações definidas em cada funcionalidade.

[10] 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 deve ser feita em qualquer outra linguagem de programação. O programa deverá compilar no GCC versão 4.8.2 ou superior para Linux.

[11] O programa deve ser acompanhado de uma **documentação externa** de, no máximo, 10 páginas. A documentação externa deve conter uma descrição em alto nível de cada algoritmo implementado para cada uma das funcionalidades [1] a [9]. Em detalhes, a documentação externa deve possuir:

- CAPA, com as seguintes informações: o nome da instituição, o nome do curso, o nome da disciplina, o nome do professor responsável, o nome do trabalho prático, o nome dos participantes e os respectivos números USP, e a data de entrega do trabalho prático.
- ÍNDICE, listando os nomes das seções que compõem o trabalho prático e as suas respectivas páginas de início.
- SEÇÕES 1 a 9: Cada uma dessas seções deve conter um algoritmo descrito em alto nível que mostra o passo-a-passo realizado para implementar a funcionalidade relacionada. Devem ser incluídas quaisquer decisões de projeto. Ou seja, a documentação referente a essas seções deve conter a descrição dos principais conceitos usados no trabalho prático, incluindo desenhos que facilitem a compreensão das estruturas de dados, as decisões de projeto e as suas justificativas, assim como qualquer outra consideração adicional assumida no desenvolvimento do trabalho prático. Também devem ser especificados nessas seções os sistemas operacionais que foram usados e como o programa deve ser compilado e executado.
- REFERÊNCIAS BIBLIOGRÁFICAS, caso necessário.

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 nas transparências 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.
- Bibliotecas necessárias para a execução do programa.
- Documentação externa em formato .pdf.

Instruções de entrega. Enviar o arquivo compactado da seguinte forma:

- e-mail: labbdciferri@gmail.com
- assunto: [Estrutura de Dados III] Trabalho Prático 2018
- corpo da mensagem: deve constar no corpo da mensagem o NUSP e nome de cada integrante do grupo. Não será atribuída nota ao aluno cujos dados não constarem no corpo da mensagem.
- documento anexado: arquivo compactado no formato .zip

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 (interna e externa) 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 interna implica que haverá uma diminuição expressiva na nota do trabalho.
- A ausência da documentação externa 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 cola, as notas dos trabalhos envolvidos serão zero (0).

Critério de avaliação dos integrantes. Podem ser incluídas uma ou mais perguntas a respeito do trabalho na prova.

Data de Entrega do Trabalho

Na data especificada na página da disciplina.

Bom Trabalho !