

**Universidade de São Paulo**  
**Instituto de Ciências Matemáticas e de Computação**  
**Departamento de Ciências de Computação**  
**Disciplina de Estrutura de Dados III (SCC0607)**

docente

Profa. Dra. Cristina Dutra de Aguiar Ciferri (cdac@icmc.usp.br)

aluno PAE

João Paulo Clarindo (jpcasantos@usp.br)

colaborador

Matheus Carvalho Raimundo (mcarvalhor@usp.br)

**Primeira Parte do Trabalho Prático**

**valor: 60%**

**Este trabalho tem como objetivo armazenar dados em um arquivo binário de acordo com uma organização de campos e registros, bem como recuperar os dados armazenados.**

*O trabalho deve ser feito em, no máximo, 3 alunos. 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.*

---

**Descrição do arquivo de dados**

---

**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.
- *numeroVertices*: indica o número de cidades diferentes que estão armazenadas no arquivo de dados. Mais detalhadamente, esse campo deve armazenar o somatório de todas as cidades de origem e de todas as cidades de destino que são diferentes entre si e que estão armazenadas no arquivo de dados. Pode assumir valores inteiros positivos, e o valor 0 deve ser usado quando não houver nenhum registro – tamanho: inteiro de 4 bytes.

- *numeroArestas*: indica o número de registros que estão armazenados no arquivo de dados. Pode assumir valores inteiros positivos, e o valor 0 deve ser usado quando não houver nenhum registro – tamanho: inteiro de 4 bytes.
- *dataUltimaCompactacao*: indica a data na qual foi feita a última compactação do arquivo de dados – tamanho: *string* de 10 bytes, no formato DD/MM/AAAA. Pode assumir os valores ‘00/00/0000’, indicando que o arquivo foi carregado com dados pela primeira vez e que nenhuma compactação foi realizada ainda, ou a data atual, para indicar que o arquivo de dados foi compactado naquela data específica.

**Representação Gráfica do Registro de Cabeçalho.** O tamanho do registro de cabeçalho é de 19 bytes, representado da seguinte forma:

1 byte	4 bytes				4 bytes				10 bytes									
<i>status</i>	<i>numeroVertices</i>				<i>numeroArestas</i>				<i>dataUltimaCompactacao</i>									
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

### Observações Importantes.

- O registro de cabeçalho 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.
- Embora em aplicações do mundo real existam cidades com mesmo nome que encontram-se em estados diferentes, nesse trabalho prático não existem cidades diferentes com o mesmo nome. O arquivo .csv com os dados de entrada já garante essa característica.

**Registros de Dados.** Deve ser considerada a *organização híbrida de campos e registros*, da seguinte forma:

- Campos de tamanho fixo e campos de tamanho variável. Para os campos de tamanho variável, deve-se usar o método *delimitador entre campos*. O delimitador deve ser representado pelo caractere ‘|’ (pipe).
- Registros de tamanho fixo.

**Observação.** Cuidado ao definir a organização do arquivo de dados. Analise os slides com título “Organização híbrida de campos e registros” disponíveis no arquivo <http://wiki.icmc.usp.br/images/3/33/SCC0215012018camposRegistros.pdf>.

---

**Descrição dos Registros de Dados.** Cada registro do arquivo de dados deve conter dados relacionados à distância em quilômetros entre duas cidades, bem como o tempo previsto para a viagem.

- Campos de tamanho fixo: 12 bytes
  - *estadoOrigem* – string – tamanho: 2 bytes (armazena a sigla do estado da cidade de origem)
  - *estadoDestino* – string – tamanho: 2 bytes (armazena a sigla do estado da cidade de destino)
  - *distancia* – inteiro – tamanho: 4 bytes (armazena a distância em quilômetros entre a cidade de origem e a cidade de destino)
- Campos de tamanho variável:
  - *cidadeOrigem* – string de tamanho variável (armazena o nome da cidade de origem)
  - *cidadeDestino* – string de tamanho variável (armazena o nome da cidade de destino)
  - *tempoViagem* – string de tamanho variável (armazena o tempo, em horas, previsto para a viagem)

Os dados são fornecidos juntamente com a especificação deste trabalho prático por meio de um arquivo .csv, sendo que sua especificação encontra-se disponível na página da disciplina. No arquivo .csv, o separador de campos é vírgula (,).

**Representação Gráfica do Registro de Dados.** O tamanho de cada registro de dados deve ser de 85 bytes, representado da seguinte forma:

2 bytes	2 bytes	4 bytes	tamanho variável	tamanho variável	tamanho variável
<i>estado Origem</i>	<i>estado Destino</i>	<i>distancia</i>	<i>cidadeOrigem</i> (+ delimitador)	<i>cidadeDestino</i> (+ delimitador)	<i>tempoViagem</i> (+ delimitador)
0	1	2	3	4	5

84

### **Observações Importantes.**

- Cada registro de dados 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.
- Deve ser feita a diferenciação entre o espaço utilizado e o lixo. Para tanto, todas as *strings* devem ser finalizadas com ‘\0’ e o lixo deve ser identificado pelo caractere '#'. Ou seja, quando sobra-se espaço no final do registro, o registro deve ser completado com lixo até o seu final.
- Os campos *estadoOrigem*, *estadoDestino*, *distância*, *cidadeOrigem*, *cidadeDestino* não aceitam valores nulos. O arquivo .csv com os dados de entrada já garante essa característica.
- O campo *tempoViagem* aceita valores nulos. O arquivo .csv com os dados de entrada já garante essa característica.
- Para os campos de tamanho fixo, os valores nulos devem ser representados da seguinte forma:
  - se o campo é inteiro ou de dupla precisão, então armazena-se o valor -1
  - se o campo é do tipo *string*, então armazena-se ‘\0#####’
- Para os campos de tamanho variável, os valores nulos devem ser representados da seguinte forma:
  - deve ser armazenado apenas o caractere referente ao delimitador entre campos.
- Não é necessário realizar o tratamento de truncamento de dados. O arquivo .csv com os dados de entrada já garante essa característica.

## Programa

---

**Descrição Geral.** Implemente um programa em C por meio do qual o usuário possa obter dados de um arquivo de entrada e gerar um arquivo binário com esses dados. O usuário deve ser capaz de realizar *inserção*, *remoção* e *atualização* de dados baseado na *abordagem estática* de registros logicamente removidos, bem como a compactação (desfragmentação) do arquivo de 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 “programaTrab1”. 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.

**Descrição Específica.** O programa deve oferecer as seguintes funcionalidades:

[1] Permita a leitura de vários registros obtidos a partir de um arquivo de entrada (arquivo no formato CSV) e a gravação desses registros em um arquivo de dados de saída. O arquivo de entrada é fornecido juntamente com a especificação do projeto, enquanto que o arquivo de dados de saída deve ser gerado como parte 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 arquivoGerado.bin

**onde:**

- arquivoEntrada.bin é um arquivo .csv que contém os valores dos campos dos registros a serem armazenados no arquivo binário.
- arquivoGerado.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.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no carregamento do arquivo.

**Exemplo de execução:**

```
./programaTrab1
1 arquivoEntrada.csv arquivoGerado.bin
usar a função binarioNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo arquivoGerado.bin
```

[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.

**Entrada do programa para a funcionalidade [2]:**

2 arquivoGerado.bin

**onde:**

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

**Saída caso o programa seja executado com sucesso:**

Cada registro deve ser mostrado em uma única linha e os seus campos devem ser mostrados de forma sequencial separado por espaço. Campos que tiverem o valor nulo não devem ser mostrados. Antes de mostrar cada registro, deve ser exibido o seu RRN (número relativo do registro).

**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** (só são mostrados apenas 2 registros):

```
./programaTrab1
2 arquivoGerado.bin
0 SP SP 243 SAO CARLOS SAO PAULO 2h32m
1 SP SP 142 SAO CARLOS CAMPINAS 1h37m
```

[3] Permita a recuperação dos dados de todos os registros que satisfaçam um critério de busca determinado pelo usuário. Por exemplo, o usuário pode solicitar a exibição de todos os registros de uma determinada *cidadeOrigem*. Qualquer campo pode ser usado como forma de busca. Campos que tiverem o valor nulo não devem ser mostrados. Ademais, 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. Os dados devem ser mostrados no mesmo formato definido para a funcionalidade [2].

**Sintaxe do comando para a funcionalidade [3]:**

3 arquivoGerado.bin NomeDoCampo valor

**onde:**

- *arquivoGerado.bin* é o arquivo binário gerado conforme as especificações descritas neste trabalho prático.

- *nomeDoCampo* é o nome do campo que encontra-se definido no arquivo de dados.
- *valor* é o valor utilizado na busca, sendo esse valor diferente de NULO. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas ("").

**Saída caso o programa seja executado com sucesso:**

Podem ser encontrados vários registros que satisfaçam à condição de busca. Cada registro deve ser mostrado em uma única linha e os seus campos devem ser mostrados de forma sequencial separado por espaço. Campos com valor nulo não devem ser mostrados. Para a manipulação de *strings* com aspas duplas, pode-se usar a função *scan\_quote\_string* disponibilizada na página da disciplina. Antes de mostrar cada registro, deve ser exibido o seu RRN (número relativo do registro).

**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** (é mostrado apenas um registro, embora a funcionalidade provida pelo programa deva exibir mais do que um registro quando for o caso):

```
./programaTrab1
3 arquivoGerado cidadeDestino "CAMPINAS"
1 SP SP 142 SAO CARLOS CAMPINAS 1h37m
```

[4] Permita a recuperação dos dados de um registro, 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. Os dados solicitados devem ser mostrados no mesmo formato definido para a funcionalidade [2].

**Sintaxe do comando para a funcionalidade [4]:**

```
4 arquivoGerado.bin  RRN
```

**Saída caso o programa seja executado com sucesso:**

Será recuperado, no máximo, 1 registro. O registro deve ser mostrado em uma única linha e os seus campos devem ser mostrados de forma sequencial separado por espaço. Campos que tiverem o valor nulo não devem mostrados. Antes de mostrar o registro, deve ser exibido o seu RRN (número relativo do registro).

**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:**

```
./programaTrab1
4 arquivoGerado.bin 1
1 SP SP 142 SAO CARLOS CAMPINAS 1h37m
```

[5] Permita a remoção lógica de registros, baseado na *abordagem estática* de registros logicamente removidos. A implementação dessa funcionalidade deve seguir estritamente a matéria apresentada em sala de aula. A marcação dos registros logicamente removidos deve ser feita inserindo-se o caractere '\*' no primeiro *byte* do registro removido. Os demais caracteres devem permanecer como estavam anteriormente. Os registros a serem removidos devem ser aqueles que satisfaçam um critério de busca determinado pelo usuário, conforme detalhado na funcionalidade [3]. Não deve ser realizado o tratamento da fragmentação interna. Também não deve ser realizado o tratamento de fragmentação externa usando a técnica de coalescimento. A funcionalidade [5] deve ser executada  $n$  vezes seguidas. Em situações nas quais um determinado critério de busca não seja satisfeito, ou seja, caso a solicitação do usuário não retorne nenhum registro a ser removido, o programa deve continuar a executar as remoções até completar as  $n$  vezes seguidas. Antes de terminar a execução da funcionalidade, deve ser utilizada a função escreverNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo binário.

**Entrada do programa para a funcionalidade [5]:**

```
5 arquivoGerado.bin n
nomeCampo1 valorCampo1
nomeCampo2 valorCampo2
...
nomeCampon valorCampon
```

**onde:**

- `arquivoGerado.bin` é um arquivo binário de entrada que segue as especificações definidas neste trabalho prático. As remoções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.

- `n` é o número de remoções a serem realizadas. Para cada remoção, deve ser informado o nome do campo a ser considerado e seu critério de busca representado pelo valor do campo, de acordo com as especificações feitas na funcionalidade [3]. Cada uma das `n` remoções deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre o nome do campo e o valor do campo. 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.

**Saída caso o programa seja executado com sucesso:**

Listar o arquivo binário `arquivoGerado.bin`.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

**Exemplo de execução:**

```
./programaTrab1
5 arquivoGerado.bin 2
cidadeDestino "CAMPINAS"
distancia 243
usar a função escreverNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo arquivoGerado.bin, o
qual foi atualizado com as remoções.
```

[6] Permita a inserção de registros adicionais, baseado na *abordagem estática* de registros logicamente removidos. A implementação dessa funcionalidade deve seguir estritamente a matéria apresentada em sala de aula. Não é necessário realizar o tratamento de truncamento de dados. Portanto, a soma dos tamanhos para os campos de tamanho variável nunca deve ultrapassar o tamanho do registro de tamanho fixo. Campos com valores nulos, na entrada da funcionalidade, devem ser identificados com NULO. A funcionalidade [6] deve ser executada  $n$  vezes seguidas. Antes de terminar a execução da funcionalidade, deve ser utilizada a função escreverNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo binário.

**Entrada do programa para a funcionalidade [6]:**

```
6 arquivoGerado.bin n
estadoOrigem1 estadoDestino1 distancia1 cidadeOrigem1 cidadeDestino1
tempoViagem1
estadoOrigem2 estadoDestino2 distancia2 cidadeOrigem2 cidadeDestino2
tempoViagem2
...
estadoOrigemn estadoDestinon distancian cidadeOrigemn cidadeDestinon
tempoViagemn
```

**onde:**

- `arquivoGerado.bin` é um arquivo binário de entrada que segue as mesmas especificações definidas nesse trabalho prático. As inserções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `n` é o número de inserções a serem realizadas. Para cada inserção, deve ser informado os valores a serem inseridos no arquivo, para os campos especificados na mesma ordem que a definida nesse trabalho prático, a saber: `estadoOrigem`, `estadoDestino`, `distancia`, `cidadeOrigem`, `cidadeDestino`, `tempoViagem`. Não existe truncamento de dados. Valores nulos devem ser identificados, na entrada da funcionalidade, por NULO. Cada uma das `n` inserções deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre os valores dos campos. Os valores dos campos do tipo *string* devem ser especificados entre aspas duplas ("").

**Saída caso o programa seja executado com sucesso:**

Listar o arquivo binário `arquivoGerado.bin`.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

**Exemplo de execução:**

```
./programaTrab2
6 arquivoGerado.bin 2
SP SP 86 "ARARAQUARA" "RIBEIRAO PRETO" "1h3m"
SP SP 92 "FRANCA" "RIBEIRAO PRETO" NULO
usar a função escreverNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo arquivoGerado.bin, o
qual foi atualizado com as inserções.
```

[7] Permita a atualização de um campo específico de um registro identificado por seu RRN. Não é necessário realizar o tratamento de truncamento de dados. Portanto, a soma dos tamanhos para os campos de tamanho variável nunca deve ultrapassar o tamanho do registro de tamanho fixo. O lixo no registro atualizado, quando ele tiver tamanho menor do que o registro antes de ser atualizado, deve permanecer com os caracteres que já estavam anteriormente. Campos a serem atualizados com valores nulos devem ser identificados, na entrada da funcionalidade, com NULO. A funcionalidade [7] deve ser executada  $n$  vezes seguidas. Em situações nas quais um determinado RRN não seja encontrado, ou seja, caso a solicitação do usuário não retorne nenhum registro a ser atualizado, o programa deve continuar a executar as atualizações até completar as  $n$  vezes seguidas. Antes de terminar a execução da funcionalidade, deve ser utilizada a função escreverNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo binário.

**Entrada do programa para a funcionalidade [7]:**

```
7 arquivoGerado.bin n
RRN nomeCampo1 valorCampo1
RRN nomeCampo2 valorCampo2
...
RRN nomeCampon valorCampon
```

**onde:**

- arquivoGerado.bin é um arquivo binário de entrada que segue as mesmas especificações definidas no primeiro trabalho prático. As atualizações a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.

- n é o número de atualizações a serem realizadas. Para cada atualização, deve ser informado o valor do RRN do registro, o campo do registro a ser alterado e o novo valor desse campo. Não existe truncamento de dados. Valores nulos devem ser identificados, na entrada da funcionalidade, por NULO. Cada uma das n atualizações deve ser especificada em uma linha diferente. Deve ser deixado um espaço em branco entre o RRN e o nome do campo, e entre o nome do campo e o valor do campo. 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.

**Saída caso o programa seja executado com sucesso:**

Listar o arquivo binário arquivoGerado.bin.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no processamento do arquivo.

**Exemplo de execução:**

```
./programaTrab2
7 arquivoGerado.bin 2
0 cidadeDestino "GUARULHOS"
2 tempoViagem NULO
usar a função escreverNaTela antes de terminar a execução da
funcionalidade, para mostrar a saída do arquivo arquivoGerado.bin, o
qual foi atualizado com as atualizações
```

[8] Permita a compactação eficiente (desfragmentação) do arquivo de dados. A compactação elimina os registros removidos, deixando no arquivo de dados somente os registros marcados como não removidos. O campo dataUltimaCompactacao deve armazenar a data na qual a compactação foi realizada. Como entrada dessa funcionalidade, tem-se um arquivo de dados binário que pode conter (ou não) registros logicamente removidos. Como saída, tem-se um arquivo de dados binário sem registros marcados como removidos. Antes de terminar a execução da funcionalidade, deve ser utilizada a função escreverNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída do arquivo binário.

**Entrada do programa para a funcionalidade [8]:**

8 arquivoGeradoEntrada.bin arquivoGerado.bin

**Saída caso o programa seja executado com sucesso:**

Listar o arquivo de saída no formato binário.

**Mensagem de saída caso algum erro seja encontrado:**

Falha no carregamento do arquivo.

**Exemplo de execução:**

```
./programaTrab1
```

```
8 arquivoGeradoEntrada.bin arquivoGerado.bin
```

```
usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo arquivoGerado.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 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. Para se fazer a busca, é possível caminhar no arquivo registro a registro, já que se sabe o tamanho do 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. 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 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 programaTrab1 *.c  
run:  
. /programaTrab1
```

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: **L31R**

---

### 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.

**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 !**