Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Organização de Arquivos (SCC0215)

Docente
Prof. MSc. Anderson Canale Garcia
andersongarcia@usp.br
Profa. Dra. Cristina Dutra de Aguiar
cdac@icmc.usp.br

Monitores
Beatriz Aimee Teixeira Furtado Braga
beatriztfb@usp.br ou telegram: @bia_aimee
Lucas Piovani
lucaspiovani@usp.br ou telegram: @lucaspiovanii
Vitor Augusto Paiva de Brito
vitor_brito@usp.br ou telegram: @vitorpbrito

Trabalho Prático 2

Este trabalho tem como objetivo indexar arquivos de dados usando um índice árvore-B e utilizar esse índice para a recuperação de dados.

O trabalho deve ser feito por, no máximo, 2 alunos da mesma turma. Os alunos devem ser os mesmos do primeiro e do segundo trabalhos práticos. Caso haja mudanças, elas devem ser informadas para a(o) docente, o aluno PAE e os monitores. 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 índice árvore-B

O índice de árvore-B com ordem *m* é definido formalmente como descrito a seguir.

1. Cada página (ou nó) do índice árvore-B deve ser, pelos menos, da seguinte forma:

   $< <C_1,P_{R1}>, <C_2,P_{R2}>, ..., <C_{q-1},P_{Rq-1}>, P_1, P_2, ..., P_{q-1}, P_q>$, onde $(q \leq m)$ e

   - Cada C_i $(1 \leq i \leq q-1)$ é uma chave de busca.
   - Cada P_{Ri} $(1 \leq i \leq q-1)$ é um campo de referência para o registro no arquivo de dados que contém o registro de dados correspondente a C_i.
   - Cada P_j $(1 \leq j \leq q)$ é um campo de referência para uma subárvore ou assume o valor -1 caso não exista subárvore (ou seja, caso seja um nó folha).

2. Dentro de cada página (ou seja, as chaves de busca são ordenadas)
   - $C_1 < C_2 < ... < C_{q-1}$.
3. Para todos os valores X da chave na subárvore apontada por P_i:
   - $C_{i-1} < X < C_i$ para $1 < i < q$
   - $X < C_i$ para $i = 1$
   - $C_{i-1} < X$ para i = q.
4. Cada página possui um máximo de m descendentes.
5. Cada página, exceto a raiz e as folhas, possui no mínimo $\lceil m/2 \rceil$ descendentes (*taxa de ocupação*).
6. A raiz possui pelo menos 2 descendentes, a menos que seja um nó folha.
7. Todas as folhas aparecem no mesmo nível.
8. Uma página não folha com k descendentes possui k-1 chaves.
9. Uma página folha possui no mínimo $\lceil m/2 \rceil - 1$ chaves e no máximo m – 1 chaves (*taxa de ocupação*).

Descrição do Registro de Cabeçalho. O registro de cabeçalho deve conter os seguintes campos:

- *status*: indica a consistência do arquivo de índice, devido à queda de energia, travamento do programa, etc. Pode assumir os valores '0', para indicar que o arquivo de índice está inconsistente, ou '1', para indicar que o arquivo de índice 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.
- *noRaiz*: armazena o RRN do nó (página) raiz do índice árvore-B. Quando a árvore-B está vazia, noRaiz = -1 – tamanho: inteiro de 4 *bytes*
- *proxRRN:* armazena o valor do próximo RRN a ser usado para conter um nó (página da árvore-B). Inicialmente, a árvore-B está vazia e, portanto, *proxRRN* = 0. Quando o primeiro nó é criado (nó folha = nó raiz), *proxRRN* = 1. Depois, quando o primeiro *split* acontece, *proxRRN* = 3. A cada nó criado da árvore-B, *proxRRN* é incrementado – tamanho: inteiro de 4 *bytes*

- *nroChaves:* armazena o número de chaves que estão indexadas pelo índice árvore-B. Inicialmente, a árvore-B está vazia e, portanto, *nroChaves* = 0. A cada nova chave inserida na árvore-B, *nroChaves* é incrementado e a cada chave removida da árvore-B, *nroChaves* é decrementado – tamanho: inteiro de 4 *bytes*

Representação Gráfica do Registro de Cabeçalho. O registro é representado da seguinte forma:

1 byte	4 bytes				4 bytes				4 bytes				demais bytes	
status	*noRaiz*				*proxRRN*				*nroChaves*				*lixo (caractere '$')*	
0	1	2	3	4	5	6	7	8	9	10	11	12	...	59

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.
- Para seguir a especificação do conceito de árvore-B, o nó da árvore-B deve obrigatoriamente ser do tamanho de uma página de disco. Entretanto, isso não será seguido neste trabalho para simplificar a quantidade de chaves de busca que são armazenadas no nó. Lembrando também que as páginas de disco têm potência de 2, o que também não será seguido neste trabalho por simplificação.
- Os bytes restantes devem ser preenchidos com lixo. O lixo é representado pelo caractere '$'.

Descrição do Registro de Dados. Deve ser considerada a seguinte organização: campos de tamanho fixo e registros de tamanho fixo. Em adição ao Item 1 da definição formal do índice árvore-B, cada nó (página) da árvore também deve armazenar dois outros campos:

- *alturaNo*: indica a altura de um nó, da seguinte forma: Os nós folhas têm altura zero; os nós do primeiro nível acima dos nós folha têm altura 1; os nós do segundo nível acima do nós folha têm altura 2; e assim sucessivamente – tamanho: inteiro de 4 *bytes*.

- *nroChaves*, indicando o número de chaves presentes no nó – tamanho: inteiro de 4 *bytes*.

A ordem da árvore-B é 4, ou seja, $m = 4$. Portanto, um nó (página) terá 3 chaves no máximo e 4 descendentes. A chave de busca é o campo *id*. Lembrando que, em aplicações reais, a ordem da árvore-B é muito maior, para acomodar mais chaves. A proposta da árvore-B é que ela seja larga e baixa, para diminuir o número de acessos a disco.

Detalhes sobre o algoritmo de inserção. Considere que deve ser implementada a rotina de *split* durante a inserção. Considere que a rotina de redistribuição durante a inserção não deve ser implementada. Considere que a distribuição das chaves de busca deve ser o mais uniforme possível, ou seja, considere que a chave de busca a ser promovida deve ser a chave que distribui o mais uniformemente possível as demais chaves entre o nó à esquerda e o novo nó resultante do particionamento. Considere também que a chave de busca a ser promovida deve ser a última chave do nó à esquerda (ou seja, o último elemento do nó existente e que está sofrendo o split é a chave promovida durante o particionamento). Quando necessário, o nó mais à direita deverá conter uma chave de busca a mais. Considere que a página sendo criada é sempre a página à direita.

Representação Gráfica de um Nó (Página/Registro de Dados) do índice. O registro é representado da seguinte forma:

4 bytes	4 bytes			4 bytes	8 bytes	4 bytes	8 bytes	4 bytes	8 bytes	4 bytes	4 bytes	4 bytes	4 bytes
altura No	*Nro Chaves*			C_1	P_{R1}	C_2	P_{R2}	C_3	P_{R3}	P_1	P_2	P_3	P_4
0 … 3	4	5	6	7	8 …								… 59

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.

- Quando um nó (página) do índice tiver chaves de busca que não forem preenchidas, a chave de busca deve ser representada pelo valor -1 e o ponteiro para o arquivo de dados deve ser representado pelo valor -1.
- O valor -1 deve ser usado para denotar que um ponteiro P_i $(1 \leq i \leq m)$ de um nó da árvore-B é nulo.

Programa

Descrição Geral. Implemente um programa em C por meio do qual o usuário possa inserir, remover e atualizar dados de arquivos binários, bem como criar índices para indexar esses arquivos. Os índices são caracterizados por serem do tipo árvore-B.

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, indica-se um campo (ou um conjunto de campos) que consiste na chave primária da tabela. Isso é realizado especificando-se a cláusula PRIMARY KEY. A funcionalidade [7] representa um exemplo de implementação de um índice árvore-B definido sobre o campo chave primária de um arquivo de dados.

Na linguagem SQL, o comando CREATE INDEX é usado para criar um índice sobre um campo (ou um conjunto de campos) de busca. A funcionalidade [7] representa um exemplo de implementação de um índice árvore-B definido sobre o campo chave primária de um arquivo de dados.

[7] Crie um arquivo de índice árvore-B para um arquivo de dados de entrada já existente, que é o arquivo de dados definido de acordo com a especificação do primeiro trabalho introdutório, e que pode conter registros logicamente removidos. O campo a ser indexado é *id*. Registros logicamente removidos presentes no arquivo de dados de entrada não devem ter suas chaves de busca correspondentes no arquivo de índice. A inserção no arquivo de índice deve ser feita um-a-um. Ou seja, para cada registro não removido presente no arquivo de dados, deve ser feita a inserção de sua chave de busca correspondente no arquivo de índice árvore-B. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. 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 de índice árvore-B.

Entrada do programa para a funcionalidade [7]:

```
7 arquivoDados.bin arquivoIndice.bin
```

onde:

- `arquivoDados.bin` é o arquivo binário que contém dados dos jogadores e que foi gerado conforme as especificações descritas no trabalho prático introdutório.
- `arquivoIndice.bin` é um arquivo binário que indexa o arquivo de dados `arquivoDados.bin` e que é gerado conforme as especificações descritas neste trabalho prático.

Saída caso o programa seja executado com sucesso:

`Listar o arquivo de índice 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
7 jogador.bin jogadorIndice.bin
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída do arquivo jogadorIndice.bin.`

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 [8] representa um exemplo de implementação do comando SELECT considerando a cláusula WHERE. Como existe um índice árvore-B definido sobre os campos dos registros, a implementação dessa funcionalidade consiste em utilizar esse índice para recuperar os dados desejados.

[8] Permita a recuperação dos dados de todos os registros que satisfaçam um critério de busca determinado pelo usuário sobre o campo *id*, usando o índice árvore-B criado na funcionalidade [7]. Note que somente o campo *id* deve ser utilizado como forma de busca, desde que o índice criado na funcionalidade [7] indexa chaves de busca desse campo. Essa funcionalidade pode retornar 0 registros (quando nenhum satisfaz ao critério de busca) ou 1 registro (desde que o campo *id* não aceita valores repetidos). A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Os dados devem ser mostrados no mesmo formato definido para a funcionalidade [3]. Registros marcados como logicamente removidos não devem ser exibidos.

Sintaxe do comando para a funcionalidade [8]:

`8 arquivoEntrada.bin arquivoIndice.bin n`

id_1 $valorID_1$

id_2 $valorID_2$

…

id_n $valorID_n$

onde:

- `arquivoEntrada.bin` é o arquivo binário de um determinado tipo, o qual foi gerado conforme as especificações descritas no trabalho prático introdutório.
- `arquivoIndice.bin` é o arquivo binário de índice árvore-B que indexa o campo *id*. Esse arquivo deve seguir as especificações definidas neste trabalho prático.
- `n` é a quantidade de vezes que a busca deve acontecer.
- `id` é o nome do campo indexado.
- `valorID` é o valor do campo *id*.

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 explicação de seu conteúdo. Deve ser deixada uma linha em branco depois de cada registro. Caso o campo seja nulo, deve ser exibido o valor SEM DADO. Também deve ser informado o número da busca que está sendo feita. 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
8 jogador.bin jogadorIndice.bin 2
1 id 261529
2 id 374857


Busca 1
* deixar uma linha em branco *
```

```
Nome do Jogador: D. FOMIN
Nacionalidade do Jogador: RUSSIA
Clube do Jogador: SEM DADO
* deixar uma linha em branco *
Busca 2
* deixar uma linha em branco *
Nome do Jogador: FIDALGO
Nacionalidade do Jogador: SPAIN
Clube do Jogador: CLUB AMERICA
```

[9] 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 *id* ou considerando os campos *nacionalidade* e *idade*. Se o critério de busca for definido em termos do campo *id* (ou qualquer combinação envolvendo esse campo) a busca deve ser realizada usando o índice criado na funcionalidade [7]. Caso contrário, a busca deve ser realizada conforme a especificação da funcionalidade [3]. Essa funcionalidade pode retornar 0 registros (quando nenhum satisfaz ao critério de busca), 1 registro (desde que o campo id não aceita valores repetidos), 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. A manipulação do arquivo de índice árvore-B deve ser feita em disco, de acordo com o conteúdo ministrado em sala de aula. Os dados devem ser mostrados no mesmo formato definido para a funcionalidade [3]. Registros marcados como logicamente removidos não devem ser exibidos. A funcionalidade [9] deve ser executada *n* vezes seguidas.

Sintaxe do comando para a funcionalidade [9]:

```
9 arquivoEntrada.bin arquivoIndice.bin n
m1 nomeCampo1 valorCampo1 ... nomeCampom1 valorCampom1
m2 nomeCampo1 valorCampo1 ... nomeCampom2 valorCampom2
...
mn nomeCampo1 valorCampo1 ... nomeCampomn valorCampomn
```

onde:

- `arquivoEntrada.bin` é o arquivo binário de um determinado tipo, o qual foi gerado conforme as especificações descritas no trabalho prático introdutório.
- `arquivoIndice.bin` é o arquivo binário de índice árvore-B que indexa o campo *id*. Esse arquivo deve seguir as especificações definidas 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 explicação de seu conteúdo. Deve ser deixada uma linha em branco depois de cada registro. Caso o campo seja nulo, deve ser exibido o valor SEM DADO. Também deve ser informado o número da busca que está sendo feita. 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
9 jogador.bin jogadorIndice.bin 2
1 id 261529
2 nacionalidade "SPAIN" idade 24
Busca 1
* deixar uma linha em branco
Nome do Jogador: D. FOMIN
Nacionalidade do Jogador: RUSSIA
Clube do Jogador: SEM DADO
* deixar uma linha em branco *
Busca 2
* deixar uma linha em branco *
Nome do Jogador: FIDALGO
Nacionalidade do Jogador: SPAIN
Clube do Jogador: CLUB AMERICA
* deixar uma linha em branco *
Nome do Jogador: OSCAR GIL
Nacionalidade do Jogador: SPAIN
Clube do Jogador: RCD ESPANYOL DE BARCELONA
* deixar uma linha em branco *
```

Na linguagem SQL, o comando INSERT INTO é usado para inserir dados em uma tabela. Para tanto, devem ser especificados os valores a serem armazenados em cada coluna da tabela, de acordo com o tipo de dado definido. A funcionalidade [10] representa exemplo de implementação do comando INSERT INTO.

[10] Permita a inserção de novos registros em um arquivo de dados de entrada, baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve ser realizada usando o conceito de *lista de registros logicamente removidos*, e deve seguir estritamente a matéria apresentada em sala de aula. A lista deve ser implementada como uma lista ordenada de forma crescente considerando os tamanhos dos registros logicamente removidos e deve ser aplicada a estratégia Best Fit. O lixo que permanece no registro logicamente removido e que é reutilizado deve ser identificado pelo caractere '$'. Na entrada desta funcionalidade, os dados são referentes aos seguintes campos, na seguinte ordem: id, idade, nomeJogador, nacionalidade, nomeClube. Campos com valores nulos, na entrada da funcionalidade, devem ser identificados com NULO. 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. A funcionalidade [10] deve ser executada *n* vezes seguidas. Antes de terminar a execução da funcionalidade, deve ser utilizada a função binarioNaTela, disponibilizada na página do projeto da disciplina, para mostrar a saída dos arquivos binários (de dados e de índice árvore-B).

Entrada do programa para a funcionalidade [10]:

```
10 arquivoDados.bin arquivoIndice.bin n
id1, idade1, nomeJogador1, nacionalidade1, nomeClube1
id2, idade2, nomeJogador2, nacionalidade2, nomeClube2
...
idn, idaden, nomeJogadorn, nacionalidaden, nomeCluben
```

onde:

- `arquivoDados.bin` é o arquivo binário que contém dados dos jogadores e que foi gerado conforme as especificações descritas no trabalho prático introdutório. As inserções a serem realizadas nessa funcionalidade devem ser feitas nesse arquivo.
- `arquivoIndice.bin` é um arquivo binário que indexa o arquivo de dados `arquivoDados.bin` e que foi gerado conforme as especificações descritas neste trabalho prático. As inserções realizadas no arquivo de dados devem ser refletidas nesse arquivo.
- `n` é o número de inserções a serem realizadas. Para cada inserção, deve ser informado os valores a serem inseridos no arquivo, considerando os seguintes campos, na seguinte ordem: id, idade, nomeJogador, nacionalidade, nomeClube. 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 de dados e o arquivo de índice 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
10 jogador.bin jogadorIndice.bin 2
261529 24 "D. FOMIN" "RUSSIA" NULO
247106 21 "A. BERNABEI" "ARGENTINA" CELTIC
```

`usar a função binarioNaTela antes de terminar a execução da funcionalidade, para mostrar a saída dos arquivos jogador.bin e jogadorIndice.bin, os quais foram atualizados frente às inserções.`

Restrições

As seguintes restrições têm que ser garantidas no desenvolvimento do trabalho.

[1] O arquivo de dados e o arquivo de índice devem ser gravados 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].

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. Assim, a forma mais simples de se fazer o arquivo makefile é:

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

Lembrando que *.c já engloba todos os arquivos .c presentes no seu 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: **SQRG**
- **Turma 2** (terça-feira): código de matrícula: **DPZ4**

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.

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!