Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Disciplina de Organização de Arquivos
Profa. Dra. Cristina Dutra de Aguiar Ciferri
PAE Anderson Chaves Carniel (Turma A)
PAE João Pedro de Carvalho Castro (Turma B)

Primeira Parte do Trabalho Prático (Parte I)
Valor: 40%

Este trabalho tem como objetivo realizar operações de inserção, remoção e atualização de dados baseadas na abordagem dinâmica de reaproveitamento de espaços de registros logicamente removidos, como como a compactação (desfragmentação do arquivo 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 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 (falso) ou 1 (verdadeiro). O valor 0 indica que o arquivo de dados está inconsistente e o valor 1 indica que o arquivo de dados está consistente – tamanho: 1 byte
- topoPilha: armazena o RRN de um registro logicamente removido, ou -1 caso não haja registros logicamente removidos – tamanho: 4 bytes

Representação Gráfica do Registro de Cabeçalho. O tamanho do registro de cabeçalho deve ser de 5 bytes, representado da seguinte forma:

Tamanho do registro de tamanho fixo: 5 bytes.

1 byte	4 bytes
status	topoPilha

Observação. 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. O tamanho do registro de tamanho fixo do registro de cabeçalho é de 5 bytes.

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 *indicador de tamanho*.
- 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 Específica para a Turma A. Cada registro do arquivo de dados deve conter dados relacionados ao Censo Escolar de 2012 – Região Sudeste do Estado de São Paulo (extraídos e processados a partir de http://dados.gov.br/dataset/instituicoes-de-ensino-basico). Essa base de dados é fornecida juntamente com a especificação deste trabalho prático por meio de um arquivo .csv, sendo que sua especificação encontra-se disponível na página da disciplina). Cada registro representa uma Instituição de Ensino Básico e contém os seguintes campos:

- Campos de tamanho fixo:
  - codEscola: código da escola - *long* ou *int* – tamanho: 4 bytes
  - dataInicio: data de início do ano letivo - *string* de 10 caracteres no formato ##/##/#### (DD/MM/AAAA, ex.: 21/12/2012) – tamanho: 10 bytes
  - dataFinal: data de final do ano letivo - *string* de 10 caracteres no formato ##/##/#### (DD/MM/AAAA, ex.: 21/12/2012) – tamanho: 10 bytes

- Campos de tamanho variável:
  - nomeEscola: nome da escola – *string*
  - municipio: município onde é localizada a escola – *string*
  - endereco: endereço da escola – *string*

Representação Gráfica do Registro de Dados para a Turma A:

Tamanho do registro de tamanho fixo: 112 bytes.

4 bytes	10 bytes	10 bytes	4 bytes	...	4 bytes	...	4 bytes	...
codEscola	dataInicio	dataFinal	indicador tamanho	nomeEscola	indicador tamanho	municipio	indicador tamanho	endereco

0	1	...	110	111

Observação. O 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. O tamanho do registro de tamanho fixo é de 112 bytes, conforme ilustrado pelos byte offsets 0, 1, ..., 110, 111.

Importante. O campo de tamanho fixo codEscola não aceita valores nulos. Os campos de tamanho fixo dataInicio e dataFinal podem assumir valores nulos. No arquivo de dados, isso deve ser representado por meio do armazenamento do valor 0000000000 (ou seja, 10 vezes o número 0). Campos de tamanho variável que sejam nulos devem ser identificados pelo tamanho 0. Ou seja, o campo referente ao indicador de tamanho desses campos deve conter o valor 0. Para tanto, considere que o número 0 não fará parte do domínio da *string*. Não é necessário realizar o tratamento de truncamento de dados.

Descrição dos Registros de Dados Específica para a Turma B. Cada registro do arquivo de dados deve conter dados relacionados ao Programa Banda Larga nas Escolas – PBLE (extraídos e processados a partir de http://dados.gov.br/dataset/pble/resource/905e1e0b-46e9-4e21-9751-b16036929629). Essa base de dados é fornecida juntamente com a especificação deste trabalho prático por meio de um arquivo .csv, sendo que sua especificação encontra-se disponível na página da disciplina). Cada registro representa uma Instituição de Ensino participante do programa e contém os seguintes campos:

- Campos de tamanho fixo:
  - codINEP: código INEP da escola - *long* ou *int* – tamanho: 4 bytes
  - dataAtiv: data de ativação - *string* de 10 caracteres no formato (DD/MM/AAAA, ex.: 21/12/2012) – tamanho: 10 bytes
  - uf: sigla UF - *string* de 2 caracteres no formato ## (ex.: SP) – tamanho: 2 bytes
- Campos de tamanho variável:
  - nomeEscola: nome da escola – *string*
  - municipio: município onde é localizada a escola – *string*
  - prestadora: nome da prestadora do serviço – *string*

Representação Gráfica do Registro de Dados para a Turma B:

Tamanho do registro de tamanho fixo: 87 bytes.

<table>
<tr><td colspan="2">4 bytes</td><td>10 bytes</td><td>2 bytes</td><td>4 bytes</td><td>...</td><td>4 bytes</td><td>...</td><td>4 bytes</td><td colspan="2">...</td></tr>
<tr><td colspan="2">codINEP</td><td>dataAtiv</td><td>uf</td><td>indicador tamanho</td><td>nomeEscola</td><td>indicador tamanho</td><td>municipio</td><td>indicador tamanho</td><td colspan="2">prestadora</td></tr>
<tr><td>0</td><td>1</td><td colspan="7">...</td><td>85</td><td>86</td></tr>
</table>

Observação. O 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. O tamanho do registro de tamanho fixo é de 87 bytes, conforme ilustrado pelos byte offsets 0, 1, ..., 85, 86.

Importante. O campo de tamanho fixo codINEP não aceita valores nulos. Os campos de tamanho fixo dataAtiv e uf podem assumir valores nulos. No arquivo de dados, valores nulos para o campo dataAtiv devem ser representados por meio do armazenamento do valor 0000000000 (ou seja, 10 vezes o número 0) e valores nulos para o campo uf devem ser representados por meio do armazenamento do valor 00 (ou seja, 2 vezes o número 0). Campos de tamanho variável que sejam nulos devem ser identificados pelo tamanho 0. Ou seja, o campo referente ao indicador de tamanho desses campos deve conter o valor 0. Para tanto, considere que o número 0 não fará parte do domínio da *string*. Não é necessário realizar o tratamento de truncamento de dados.

Programa

Descrição Geral. Implemente um programa em C que ofereça uma interface, via linha de comando, por meio da qual o usuário possa realizar *inserção*, *remoção* e *atualização* de dados baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos, bem como a compactação (desfragmentação) do arquivo de dados. Deve-se levar em consideração a descrição e a organização do arquivo de dados especificados 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 "programaTrab1" (entretanto, o grupo pode definir qualquer outro nome para o programa). No final da especificação desse trabalho, é disponibilizado um programa exemplo para mostrar como é realizada a captura de parâmetros via linha de comando e como os dados devem ser enviados para a saída padrão. 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 (ou seja, um inteiro de 1 a 9), conforme especificado a seguir.

Descrição Específica para as Turmas A e B. 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 será fornecido juntamente com a especificação do projeto, enquanto que o arquivo de dados de saída deverá ser gerado como parte deste trabalho prático.

Sintaxe do comando para a funcionalidade [1]:

```
./programaTrab1 1 'arquivo.csv'
```

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo carregado.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no carregamento do arquivo.
```

Exemplo de execução:

```
./programaTrab1 1 'arquivo.csv'
Arquivo carregado.
```

[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.

Sintaxe do comando para a funcionalidade [2]:

```
./programaTrab1 2
```

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. Para os campos com tamanho variável, mostre também seu tamanho em bytes.
```

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 para a Turma A (são mostrados apenas 2 registros)**:**

```
./programaTrab1 2
35000012 01/02/2012 21/12/2012 24 AYRES DE MOURA PROFESSOR 9 SAO PAULO 17 RUA ARTUR ORLANDO
35000024 01/02/2012 21/12/2012 25 GAVIAO PEIXOTO BRIGADEIRO 9 SAO PAULO 11 RUA MOGEIRO
```

Exemplo de execução para a Turma B (são mostrados apenas 2 registros)**:**

```
./programaTrab1 2
31031917 18/09/2009 MG 20 EM PERCILIA LEONARDO 7 ARAUJOS 4 CTBC
31031984 22/03/2011 MG 14 EE JOSE MANOEL 7 ARAUJOS  4 CTBC
```

[3] Permita a recuperação dos dados de todos os registros que satisfaçam um critério de busca determinado pelo usuário. Por exemplo, para a Turma A, o usuário pode solicitar a exibição de todos os registros de um determinado *municipio*, enquanto que para a Turma B, o usuário pode solicitar a exibição de todos os registros que possuem um determinado *codINEP*. Os dados solicitados devem ser mostrados no mesmo formato definido para a funcionalidade [2].

Sintaxe do comando para a funcionalidade [3]:

```
./programaTrab1 3 'NomeDoCampo' valor
```

Observação: caso o valor seja uma *string*, ela deve estar entre aspas simples (por exemplo, 'SAO PAULO').

Saída caso o programa seja executado com sucesso:

```
Cada registro deve ser mostrado em uma única linha e os seus campos devem ser mostrado de forma sequencial separado por espaço. Para os campos com tamanho variável, mostre também seu tamanho em bytes.
```

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 para a Turma A (é mostrado apenas o primeiro registro que satisfez à busca, embora a funcionalidade provida pelo programa deva exibir mais do que um registro quando for o caso)**:**

```
./programaTrab1 3 'codEscola' 35000024
35000024 01/02/2012 21/12/2012 25 GAVIAO PEIXOTO BRIGADEIRO 9 SAO PAULO 11 RUA MOGEIRO
```

Exemplo de execução para a Turma B (é mostrado apenas o primeiro registro que satisfez à busca, embora a funcionalidade provida pelo programa deva exibir mais do que um registro quando for o caso)**:**

```
./programaTrab1 3 'nomeEscola' 'EE JOSE MANOEL'
31031984 22/03/2011 MG 14 EE JOSE MANOEL 7 ARAUJOS  4 CTBC
```

[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]:

```
./programaTrab1 4 RRN
```

Saída caso o programa seja executado com sucesso:

O registro deve ser mostrado em uma única linha e os seus campos devem ser mostrado de forma sequencial separado por espaço. Para os campos com tamanho variável, mostre também seu tamanho em bytes.

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 para a Turma A:

```
./programaTrab1 4 1
35000012 01/02/2012 21/12/2012 24 AYRES DE MOURA PROFESSOR 9 SAO PAULO 17 RUA ARTUR ORLANDO
```

Exemplo de execução para a Turma B:

```
./programaTrab1 4 1
31031917 18/09/2009 MG 20 EM PERCILIA LEONARDO 7 ARAUJOS 4 CTBC
```

[5] Permita a remoção lógica de registros, baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve ser realizada utilizando-se o conceito de *pilha*, e deve seguir estritamente a matéria apresentada em sala de aula. Cada registro a ser removido deve ser identificado a partir do RRN fornecido como entrada.

Sintaxe do comando para a funcionalidade [5]:

```
./programaTrab1 5 RRN
```

Mensagem de saída caso o programa seja executado com sucesso:

```
Registro removido com sucesso.
```

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 5 12
Registro removido com sucesso.
```

[6] Permita a inserção de registros adicionais, baseado na *abordagem dinâmica* de reaproveitamento de espaços de registros logicamente removidos. A implementação dessa funcionalidade deve ser realizada utilizando-se o conceito de *pilha*, e deve seguir estritamente a matéria apresentada em sala de aula.

Sintaxe do comando para a funcionalidade [6]:

```
./programaTrab1 6 valorCampo1 valorCampo2 valorCampo3 valorCampo4 valorCampo5 valorCampo6
```

Observações: caso o valor de um campo seja uma *string*, ela deve estar entre aspas simples (por exemplo, 'SAO PAULO'). Caso o valor seja *null*, ele deve ser identificado como 0 para os campos de tamanho fixo e como '' para os campos de tamanho variável. 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.

Mensagem de saída caso o programa seja executado com sucesso:

```
Registro inserido com sucesso.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento do arquivo.
```

Exemplo de execução para a Turma A:

```
./programaTrab1 6 49678012 0 0 'EE DISCIPLINA' 'SAO CARLOS' ''
Registro inserido com sucesso.
```

Exemplo de execução para a Turma B:

```
./programaTrab1 6 31891919 18/01/2018 SP 'EE DISCIPLINA' 'RUA INPE' ''
Registro inserido com sucesso.
```

[7] Permita a atualização de todos os campos de um registro identificado por seu RRN.

Sintaxe do comando para a funcionalidade [7]:

```
./programaTrab1 7 RRN valorCampo1 valorCampo2 valorCampo3 valorCampo4 valorCampo5 valorCampo6
```

Observações: caso o valor de um campo seja uma *string*, ela deve estar entre aspas simples (por exemplo, 'SAO PAULO'). Caso o valor seja *null*, ele deve ser identificado como 0 para os campos de tamanho fixo e como '' para os campos de tamanho variável. 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.

Mensagem de saída caso o programa seja executado com sucesso:

```
Registro alterado com sucesso.
```

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 para a Turma A:

```
./programaTrab1 7 1 35000012 0 0 'EE DISCIPLINA' 'SAO CARLOS' ''
Registro alterado com sucesso.
```

Exemplo de execução para a Turma B:

```
./programaTrab1 7 1 31031917 18/01/2018 SP 'EE DISCIPLINA' 'RUA INPE' ''
Registro alterado com sucesso.
```

[8] Permita a compactação eficiente (desfragmentação) do arquivo de dados.

Sintaxe do comando para a funcionalidade [8]:

```
./programaTrab1 8
```

Mensagem de saída caso o programa seja executado com sucesso:

```
Arquivo de dados compactado com sucesso.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento do arquivo.
```

Exemplo de execução:

```
./programaTrab1 8
Arquivo de dados compactado com sucesso.
```

[9] Permita a recuperação dos RRNs da pilha de registros logicamente removidos.

Sintaxe do comando para a funcionalidade [9]:

```
./programaTrab1 9
```

Saída caso o programa seja executado com sucesso:

```
Os RRNs devem ser mostrados na ordem de seu desempilhamento, e devem ser separados por um espaço.
```

Mensagem de saída caso não existam registros logicamente removidos:

```
Pilha vazia.
```

Mensagem de saída caso algum erro seja encontrado:

```
Falha no processamento do arquivo.
```

Exemplo de execução:

```
./programaTrab1 9
1 10 25 8
```

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] 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. Quaisquer decisões de projeto. 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: [Organização de Arquivos] Trabalho Prático 2018; Turma X; Parte Y
- 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 !