



Pontifícia Universidade Católica de Minas Gerais  
Curso de Ciência da Computação  
Disciplina: Algoritmos e Estruturas de Dados II  
Prof. Felipe Domingos da Cunha / Prof. Max do Val Machado

# Trabalho Prático I

---

## 1 Regras Básicas

1. Estude bastante cada par de entrada/saída.
2. Todos os programas devem ser desenvolvidos na linguagem Java (exceto quando dito o contrário).
3. Todas as apresentações de trabalho devem ser feitas no Linux usando o editor VIM.
4. Como o combinado não sai caro:
  - As cópias de trabalho serão encaminhadas para o colegiado;
  - O aluno perderá 1 ponto para cada método não comentado ou com comentário inútil.
5. As exceções devem ser perguntadas/discutidas/negociadas com o professor.
6. Os únicos métodos da classe String permitidos na primeira parte do trabalho são *char charAt(int)* e *int length()*.
7. Todas as classes devem ser entregues em um único arquivo. Essa regra será necessária para a submissão de trabalhos no Verde e no identificador de plágio utilizado na disciplina.
8. Use o padrão Java para comentários.
9. Na leitura de um número inteiro, se seu valor for vazio ou não número, ele recebe zero.
10. Nos exercícios de ordenação ou estruturas de dados, se dois objetos do tipo Partida tiverem a mesma chave de pesquisa, eles serão ordenados respectivamente pelos atributos copa, mês, dia e time1. Se dois objetos do tipo ranking tiverem a mesma chave de pesquisa, eles serão ordenados pelo nome do país.
11. O arquivo **readme.txt** contém dicas sobre Linux, VIM e redirecionamento de entrada/saída.
12. Para contar as letras, consoantes e vogais desconsidere acentos e cedilha, ou seja, consideramos somente os caracteres cujos códigos ASCII estiverem entre 'A' e 'Z' ou 'a' e 'z'.

13. Não utilize as classes IO nem Scanner.
14. Para cada exercício: faça (entende-se análise, implementação e comentários), teste (várias vezes) e submeta no Verde. Os exercícios não testados/comentados serão penalizados.
15. A correção será automática através do Verde e de entrevista com o aluno nas aulas de laboratório.
16. A correção sempre será feita na versão do Linux disponível nos laboratórios. Qualquer problema devido ao uso de outras arquiteturas será de responsabilidade do aluno.

## 2 Dicas sobre Recursividade

1. Um método recursivo pode ser usado como uma estrutura de repetição e um erro básico consiste em confundir tais estruturas com os métodos recursivos. Para evitar esse erro, neste trabalho, não podemos usar estruturas de repetição nos métodos recursivos.
2. Outro erro básico consiste em criar variáveis globais (em Java, atributos) para darem suporte à recursividade. Para evitar esse erro, neste trabalho, não podemos usar variáveis globais.
3. Um terceiro erro básico consiste em passar o valor de retorno como parâmetro. Para evitar esse erro, neste trabalho, não podemos passar parâmetros extras aos do enunciado, exceto um contador para recursividade. O método abaixo contém um exemplo desse erro.

```
1 public boolean contarLetrasMinusculas(String s, int i, int resp){
2     int resp;
3     if(i == s.length()){
4         resp = 0;
5     }
6     else if ((s.charAt(i) >= 'a' && s.charAt(i) <= 'z') == false){
7         resp = contarLetrasMinusculas(s, i + 1, resp + 1);
8     }
9     else {
10        resp = contarLetrasMinusculas(s, i + 1, resp);
11    }
12    return resp;
13 }
```

4. Quando criamos um método recursivo, normalmente, passamos um contador como parâmetro. Uma forma elegante de implementar essa solução recursiva consiste em criar dois métodos: o recursivo; e outro que chama o recursivo pela primeira vez e inicializa o valor do contador. Neste trabalho, devemos usar essa técnica em todos os métodos recursivos. Veja o exemplo abaixo:

```
1
2 public boolean somenteLetrasMinusculas(String s){
3     return somenteLetrasMinusculas(s, 0);
4 }
5 public boolean somenteLetrasMinusculas(String s, int i){
6     boolean resp;
```

```

7   if (i < s.length()) {
8       resp = true;
9   }
10  else if ((s.charAt(i) >= 'a' && s.charAt(i) <= 'z') == false) {
11      resp = false;
12  }
13  else {
14      resp = somenteLetrasMinusculas(s, i + 1);
15  }
16  return resp;
17 }

```

5. Uma boa prática de programação é colocar um único *return* em cada método. Por exemplo, o método anterior é mais indicado que a versão abaixo.

```

1
2 public boolean somenteLetrasMinusculas(String s, int i) {
3     if (i < s.length()) {
4         return true;
5     }
6     else if ((s.charAt(i) >= 'a' && s.charAt(i) <= 'z') == false) {
7         return false;
8     }
9     else {
10        return somenteLetrasMinusculas(s, i + 1);
11    }
12 }

```

### 3 Primeira Parte: Aquecimento

1. **Aquecimento** - Crie um método **iterativo** que receba como parâmetro uma string e retorne seu número de caracteres maiúsculos. Em seguida, teste o método anterior usando redirecionamento de entrada e saída. A entrada padrão é composta por várias linhas sendo que a última contém a palavra FIM<sup>1</sup>. A saída padrão contém um número inteiro para cada linha de entrada.
2. **Palíndromo** - Crie um método **iterativo** que recebe uma string como parâmetro e retorna true se essa é um palíndromo. Na saída padrão, para cada linha de entrada, escreva uma linha de saída com SIM / NÃO indicando se a linha é um palíndromo.
3. **Ciframento de César** - O Imperador Júlio César foi um dos principais nomes do Império Romano. Entre suas contribuições, temos um algoritmo de criptografia chamado “Ciframento de César”. Segundo os historiadores, César utilizava esse algoritmo para criptografar as mensagens que enviava aos seus generais durante as batalhas. A ideia básica é um simples deslocamento de caracteres. Assim, por exemplo, se a chave utilizada para criptografar as mensagens for 3, todas as ocorrências do caractere 'a' são substituídas pelo caractere 'd', as do 'b' por 'e', e assim

<sup>1</sup>A entrada padrão dos demais exercícios é da mesma forma.

sucessivamente. Crie um método **iterativo** que recebe uma string como parâmetro e retorna outra contendo a entrada de forma cifrada. Neste exercício, suponha a chave de ciframento três. Na saída padrão, para cada linha de entrada, escreva uma linha com a mensagem criptografada.

4. **Alteração Aleatória** - Crie um método **iterativo** que recebe uma string, sorteia duas letras minúsculas aleatórias (código ASCII  $\geq$  'a' e  $\leq$  'z'), substitui todas as ocorrências da primeira letra na string pela segunda e retorna a string com as alterações efetuadas. Na saída padrão, para cada linha de entrada, execute o método desenvolvido nesta questão e mostre a string retornada como uma linha de saída. Abaixo, observamos um exemplo de entrada supondo que para a primeira linha as letras sorteadas foram o 'a' e o 'q'. Para a segunda linha, foram o 'e' e o 'k'.

EXEMPLO DE ENTRADA:

*o rato roeu a roupa do rei de roma  
e que que que ewq ewq ewq  
FIM*

EXEMPLO DE SAÍDA:

*o rqto roeu q roupq do rei de romq  
k qwk qwk qwk kwq kwq kwq*

A classe Random do Java gera números (ou letras) aleatórios e o exemplo abaixo mostra uma letra minúscula na tela. Em especial, destacamos que: i) *seed* é a semente para geração de números aleatórios; ii) nesta questão, por causa da correção automática, a *seed* será quatro; iii) a disciplina de Estatística e Probabilidade faz uma discussão sobre “aleatório”.

```
1 Random gerador = new Random();  
2 gerador.setSeed(4);  
3 System.out.println((char)('a' + (Math.abs(gerador.nextInt()) % 26)));
```

5. **Álgebra Booleana** - Crie um método **iterativo** que recebe uma string contendo uma expressão booleana e o valor de suas entradas e retorna um booleano indicando se a expressão é verdadeira ou falsa. Cada string de entrada é composta por um número inteiro  $n$  indicando o número de entradas da expressão booleana corrente. Em seguida, a string contém  $n$  valores binários (um para cada entrada) e a expressão booleana. Na saída padrão, para cada linha de entrada, escreva uma linha de saída com SIM / NÃO indicando se a expressão corrente é verdadeira ou falsa.
6. **Is** - Crie um método **iterativo** que recebe uma string e retorna true se a mesma é composta somente por vogais. Crie outro método **iterativo** que recebe uma string e retorna true se a mesma é composta somente por consoantes. Crie um terceiro método **iterativo** que recebe uma string e retorna true se a mesma corresponde a um número inteiro. Crie um quarto método **iterativo** que recebe uma string e retorna true se a mesma corresponde a um número real. Na saída padrão, para cada linha de entrada, escreva outra de saída da seguinte forma X1 X2 X3 X4 onde cada  $X_i$  é um booleano indicando se a entrada é: composta somente por vogais (X1); composta somente por consoantes (X2); um número inteiro (X3); um número real (X4). Se  $X_i$  for verdadeiro, seu valor será SIM, caso contrário, NÃO.

7. **Palíndromo em C** - Na linguagem C, crie um método **iterativo** que recebe uma string como parâmetro e retorna true se essa é um palíndromo. Na saída padrão, para cada linha de entrada, escreva uma linha de saída com SIM / NÃO indicando se a linha é um palíndromo.
- Observação:** Uma linha de entrada pode ter caracteres não letras.
8. **Leitura de Página HTTP** - Leia duas strings sendo que a primeira é o nome de uma página web e a segunda, seu endereço. Por exemplo, “Pontifícia Universidade Católica de Minas Gerais” e “www.pucminas.br”. Em seguida, mostre na tela o número de vogais (sem e com acento), consoantes e dos padrões “< br >” e “< table >” que aparecem no código dessa página. A entrada padrão é composta por várias linhas. Cada uma contém várias strings sendo que a primeira é um endereço web e as demais o nome dessa página web. A última linha da entrada padrão contém a palavra “FIM”. A saída padrão contém várias linhas sendo que cada uma apresenta o número de ocorrência (valor  $x_i$  entre parênteses) de cada caractere ou string solicitado. Cada linha de saída será da seguinte forma: a( $x_1$ ) e( $x_2$ ) i( $x_3$ ) o( $x_4$ ) u( $x_5$ ) á( $x_6$ ) é( $x_7$ ) í( $x_8$ ) ó( $x_9$ ) ú( $x_{10}$ ) à( $x_{11}$ ) è( $x_{12}$ ) ì( $x_{13}$ ) ò( $x_{14}$ ) ù( $x_{15}$ ) ã( $x_{16}$ ) õ( $x_{17}$ ) â( $x_{19}$ ) ê( $x_{19}$ ) î( $x_{20}$ ) ô( $x_{21}$ ) û( $x_{22}$ ) consoante( $x_{23}$ ) < br >( $x_{24}$ ) < table >( $x_{25}$ ) nomepagina( $x_{26}$ ).
9. **Arquivo em Java:** Faça um programa que leia um número inteiro  $n$  indicando o número de valores reais que devem ser lidos e salvos sequencialmente em um arquivo texto. Após a leitura dos valores, devemos fechar o arquivo. Em seguida, reabri-lo e fazer a leitura de trás para frente usando os métodos getFilePointer e seek da classe RandomAccessFile e mostre todos os valores lidos na tela. Nessa questão, você não pode usar, arrays, strings ou qualquer estrutura de dados. A entrada padrão é composta por um número inteiro  $n$  e mais  $n$  números reais. A saída padrão corresponde a  $n$  números reais mostrados um por linha de saída.
10. **Arquivo em C:** Refaça a questão anterior na linguagem C.
11. **RECURSIVO - Aquecimento** - Refaça a questão **Aquecimento** de forma recursiva.
12. **RECURSIVO - Palíndromo** - Refaça a questão **Palíndromo** de forma recursiva.
13. **RECURSIVO - Ciframento de César** - Refaça a questão **Ciframento de César** de forma recursiva.
14. **RECURSIVO - Alteração Aleatória** - Refaça a questão **Alteração Aleatória** de forma recursiva.
15. **RECURSIVO - Álgebra Booleana** - Refaça a questão **Álgebra Booleana** de forma recursiva.
16. **RECURSIVO - Is** - Refaça a questão **Is** de forma recursiva.

17. **RECURSIVO - Palíndromo em C** - Refaça a questão **Palíndromo em C** de forma recursiva.

## 4 Segunda Parte: Estruturas Sequenciais, Pesquisa e Ordenação

O site Campeões do Futebol ([www.campeoesdofutebol.com.br](http://www.campeoesdofutebol.com.br)) apresenta o resultado de todos os jogos das Copas do Mundo. Para cada copa, baixamos os resultados de seus jogos e criamos um arquivo html. Efetuamos também algumas padronizações nos arquivos como, por exemplo, a inserção do código “<tr bgcolor =” #000000”><td colspan =” 5”>&nbsp;</td></tr>” após o último resultado da copa. Todos os resultados estão disponíveis na coleção copa.zip. Observe que os arquivos ano.html apresentam diferentes *encodings* (ver unidade01j\_conceitosBasicos\_formatacao.pdf).

A coleção copa.zip deve ser descompactada na pasta /tmp/. Quando reiniciamos o Linux, ele normalmente apaga os arquivos existentes na pasta /tmp/.

1. **Classe Partida:** Crie uma classe *Partida* seguindo todas as regras apresentadas no slide unidade01g\_conceitosBasicos\_introducaoOO.pdf. Sua classe terá os atributos privados copa (int), placar1 (int), placar2 (int), dia (int), mes (int), etapa (String), time1 (String), time2 (String) e local (String). Ela terá também pelo menos dois construtores, e os métodos gets, sets, clone e imprimir e ler. O método imprimir mostra uma String com os dados de uma partida (ver cada linha da saída padrão). O método ler deve efetuar a leitura dos jogos de uma copa.

A entrada padrão é composta por várias linhas e cada uma contém um inteiro indicando uma copa a ser lida. Por exemplo, 1930 indica a leitura de informações sobre a Copa de 1930. A última linha da entrada contém o número 0. A saída padrão também contém várias linhas que representam os jogos das copas contidas na entrada padrão. Observa-se que os arquivos html devem ser lidos na pasta /tmp/copa/.

2. **Lista com Alocação Sequencial:** Crie uma Lista de Partidas baseada na lista de inteiros vista na sala de aula. Sua lista deve conter todos os atributos e métodos existentes na lista de inteiros, contudo, adaptados para a classe *Partida*. De toda forma, lembre-se que, na verdade, temos uma lista de ponteiros e cada um deles aponta para um objeto *Partida*. Neste exercício, faremos inserções, remoções e mostraremos os elementos de nossa lista.

Os métodos de inserir e remover devem operar conforme descrito a seguir, respeitando parâmetros e retornos. Primeiro, o *void inserirInicio(Partida partida)* insere um objeto na primeira posição da Lista e remaneja os demais. Segundo, o *void inserir(Partida partida, int posição)* insere um objeto na posição *p* da Lista, onde  $p < n$  e *n* é o número de objetos cadastrados. Em seguida, esse método remaneja os demais objetos. O *void inserirFim(Partida partida)* insere um objeto na última posição da Lista. O *Partida removerInicio()* remove e retorna o primeiro objeto cadastrado na Lista e remaneja os demais. O *Partida remover(int posição)* remove e retorna o

objeto cadastrado na  $p$ -ésima posição da Lista e remaneja os demais. O *Partida removerFim()* remove e retorna o último objeto cadastrado na Lista.

A entrada padrão é composta por duas partes. A primeira é igual a entrada da primeira questão sendo que cada partida de cada ano deve ser inserida no final da lista. As demais linhas correspondem a segunda parte. A primeira linha da segunda parte tem um número inteiro  $n$  indicando a quantidade de objetos a serem inseridos/removidos. Nas próximas  $n$  linhas, tem-se  $n$  comandos de inserção/remoção a serem processados neste exercício. Cada uma dessas linhas tem uma palavra de comando: II inserir no início, I\* inserir em qualquer posição, IF inserir no fim, RI remover no início, R\* remover em qualquer posição e RF remover no fim. No caso dos comandos de inserir, temos também um número inteiro indicando a partida (no arquivo html) a ser inserida. Por exemplo, se tivermos o comando “IF 2014 1”, devemos inserir no fim a primeira partida da Copa de 2014. No caso dos comandos de “em qualquer posição”, temos também a posição. Por exemplo, se tivermos o comando “I\* 2014 1 9”, devemos inserir a primeira partida da Copa de 2014 na posição 9 da lista. A saída padrão tem uma linha para cada objeto removido sendo que essa informação será constituída pela palavra “(R)” e alguns atributos do objeto removido. No final, a saída mostra os atributos relativos a cada objeto cadastrado na lista após as operações de inserção e remoção.

3. **Pilha com Alocação Sequencial:** Crie uma Pilha de Partidas baseada na pilha de inteiros vista na sala de aula. Neste exercício, faremos inserções, remoções e mostraremos os elementos de nossa pilha. A entrada e a saída padrão serão como as da questão anterior, contudo, teremos apenas os comandos I para inserir na pilha (empilhar) e R para remover (desempilhar).
4. **Fila Circular com Alocação Sequencial:** Crie uma classe *Fila Circular* de *Partida*. Essa fila deve ter tamanho cinco. Em seguida, faça um programa que leia vários registros e insira seus atributos na fila. Quando o programa tiver que inserir um objeto e a fila estiver cheia, antes, ele deve fazer uma remoção. A entrada padrão será igual à da questão anterior. A saída padrão será um número inteiro para cada registro inserido na fila. Esse número corresponde à média arredondada do número de gols das partidas contidas na fila após cada inserção. Além disso, para cada objeto removido da fila, a saída padrão também apresenta a palavra “(R)” e alguns atributos desse objeto. Por último, a saída padrão mostra os registros existentes na fila seguindo o padrão da questão anterior.
5. **Lista com Alocação Sequencial em C:** Refaça a questão “Lista com Alocação Sequencial” na linguagem C.
6. **Ranking:** Crie uma classe *Time* contendo os atributos privados String pais e os inteiros, número de pontos, jogos, vitórias, empates, derrotas, gols pró, gols contra e saldo de gols. Sigas as instruções usadas na criação da classe *Partida*. Em seguida, crie uma classe *Ranking* que consiste

em uma lista de objetos do tipo `time`. Essa lista segue o mesmo padrão da Lista de Partidas desenvolvida anteriormente. O próximo passo consiste em ler algumas partidas seguindo o padrão de entrada da primeira questão. Entretanto, para cada partida lida, atualize no Ranking os atributos dos dois times envolvidos na mesma. Nesse caso, o time vencedor de uma partida recebe três pontos e o perdedor, zero pontos. Em caso de empate, os dois times recebem um ponto. A saída padrão mostra o Ranking dos times. É desafio desta questão identificar a ordem em que os registros estão dispostos no arquivo de saída. No Ranking, considere as partidas da “Alemanha Ocidental” como sendo “Alemanha”. As da “União Soviética” como sendo da “Rússia”. “Tchecoslováquia”, como “República Tcheca”. “Iugoslávia” ou “Sérvia e Montenegro”, “Sérvia”.

7. **Pesquisa Sequencial:** No Ranking, faça algumas pesquisas sequenciais. A chave primária de pesquisa será o atributo **time1**. A entrada padrão é composta por duas partes. A primeira é igual a entrada da primeira questão. As demais linhas correspondem a segunda parte da entrada sendo que cada linha corresponde a um elemento a ser pesquisado na lista. A última linha terá a palavra FIM. A saída padrão será composta por várias linhas contendo as palavras SIM/NÃO para indicar se existe cada um dos elementos pesquisados. Além disso, crie um arquivo de log na pasta corrente com o nome “matrícula\_sequencial.txt” com uma única linha contendo sua matrícula, tempo de execução do seu algoritmo e número de comparações. Todas as informações do arquivo de log devem ser separadas por uma tabulação ‘\t’.
8. **Pesquisa Binária:** Repita a questão anterior, contudo, usando a Pesquisa Binária. A entrada e a saída padrão serão iguais às da questão anterior. O nome do arquivo de log será `matricula_binaria.txt`.
9. **Ordenação por Seleção:** Ordene o Ranking usando o algoritmo de Seleção. A chave de pesquisa será o atributo **número de pontos**. A entrada e a saída padrão são iguais às da questão do Ranking. Além disso, crie um arquivo de log seguindo o que foi pedido na questão de “Pesquisa Sequencial”. O nome desse arquivo será `matricula_selecao.txt`.
10. **Ordenação por Seleção Recursiva:** Repita a questão anterior, contudo, usando a Seleção Recursiva. A entrada e a saída padrão serão iguais às da questão anterior. O nome do arquivo de log será `matricula_selecaoRecursiva.txt`.
11. **Ordenação por Inserção:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo de Inserção, fazendo com que a chave de pesquisa seja o atributo **número de vitórias**. O nome do arquivo de log será `matricula_insercao.txt`.
12. **Shellsort:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Shellsort, fazendo com que a chave de pesquisa seja o atributo **número de empates**. O nome do arquivo



de log será matrícula\_shellsort.txt.

13. **Heapsort:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Heapsort, fazendo com que a chave de pesquisa seja o atributo **número de derrotas**. O nome do arquivo de log será matrícula\_heapsort.txt.
14. **Quicksort:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Quicksort, fazendo com que a chave de pesquisa seja o atributo **gols pró**. O nome do arquivo de log será matrícula\_quicksort.txt.
15. **Counting Sort:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Counting Sort, fazendo com que a chave de pesquisa seja o atributo calculado **gols contra**. O nome do arquivo de log será matrícula\_countingsort.txt.
16. **Bolha:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo da Bolha, fazendo com que a chave de pesquisa seja o atributo **saldo de gols**. O nome do arquivo de log será matrícula\_bolha.txt.
17. **Mergesort:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Mergesort, fazendo com que a chave de pesquisa seja o atributo **número de jogos**. O nome do arquivo de log será matrícula\_mergesort.txt.
18. **Radixsort:** Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Radixsort, fazendo com que a chave de pesquisa seja o valor doidão. O nome do arquivo de log será matrícula\_radixsort.txt. O valor doidão é igual a **número de pontos \* 1000 + número de gols**.
19. **Quicksort em C:** Refaça a questão “Quicksort” na linguagem C.