Questão 1) No contexto de estrutura de dados, um vetor é uma sequência contígua de elementos armazenados em memória, onde cada elemento é acessado através de um índice numérico. Também conhecido pelos termos arranjo e *array*, ele é uma das estruturas de dados mais simples, sendo amplamente utilizado em algoritmos de ordenação, pesquisa e manipulação de dados.

Algumas aplicações podem organizar internamente os elementos de um vetor segundo alguma propriedade. Por exemplo, podemos organizar os elementos S_i de um vetor de forma que

$$S_i \leq S_{\lfloor (i-1)/2 \rfloor}, para\ 0 < i \leq n-1,$$

onde **i** representa o índice do elemento no vetor e **n** representa o número de elementos no vetor. Um vetor organizado de modo a respeitar essa propriedade é conhecido como *heap* máxima. Em uma *heap* podemos criar um relacionamento entre duas posições de um vetor, chamando-os de pai e filho, criando assim uma abstração que nos permite interpretar todo o vetor como uma árvore binária totalmente preenchida em todos os seus níveis exceto, possivelmente, o último. O trecho de código abaixo implementa, na linguagem de programação C, uma *heap* máxima:

```
#include<stdio.h>
#include<limits.h>

int parent(int i) { return (i-1)/2; }

int left(int i) { return 2*i+1; }

int right(int i) { return 2*i+2; }

void swap (int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b= temp;
}

void maxHeapify(int A[ ], int n, int i) {
    int l = left(i);
    int r = right(i);
    int largest = INT_MIN;
    if (l <= (n-1) && A[l] > A[i]) largest = l;
    else largest = i;
    if (r <= (n-1) && A[r] > A[largest]) largest = r;
    if (largest != i) {
        swap(&A[i], &A[largest]);
        maxHeapify(A, n, largest);
    }
}
```

```
void buildMaxHeap(int A[ ], int n) {
   int i;
   for(i = n/2-1;i >= 0;i--)
      maxHeapify(A, n, i);
}
```

a) **(0,5 ponto)** A partir da análise do código acima, podemos considerar que o vetor **A**={16,14,10,5,8,7,2,1,9,4} representa uma *heap* máxima obtida após o retorno da execução da função **buildMaxHeap**? Justifique sua resposta.

b) **(1,5 ponto)** Escreva uma função chamada **heapsort** na linguagem de programação C que ordene em ordem crescente um vetor de inteiros **A**. A função **heapsort** deve obrigatoriamente ser implementada a partir de chamadas às funções **buildMaxHeap**, **swap** e **maxHeapify**. Assuma que tanto o vetor **A** quanto o seu tamanho **n** (do tipo inteiro) sejam passados como parâmetros para a função **heapsort**. Considere, adicionalmente, que a ordenação seja local, ou seja, ocorra no próprio vetor **A**.

Questão 2) Considere:

i. A existência de um servidor HTTP localizado em *"http://localhost"*, na porta 80, que seja capaz de processar arquivos PHP.

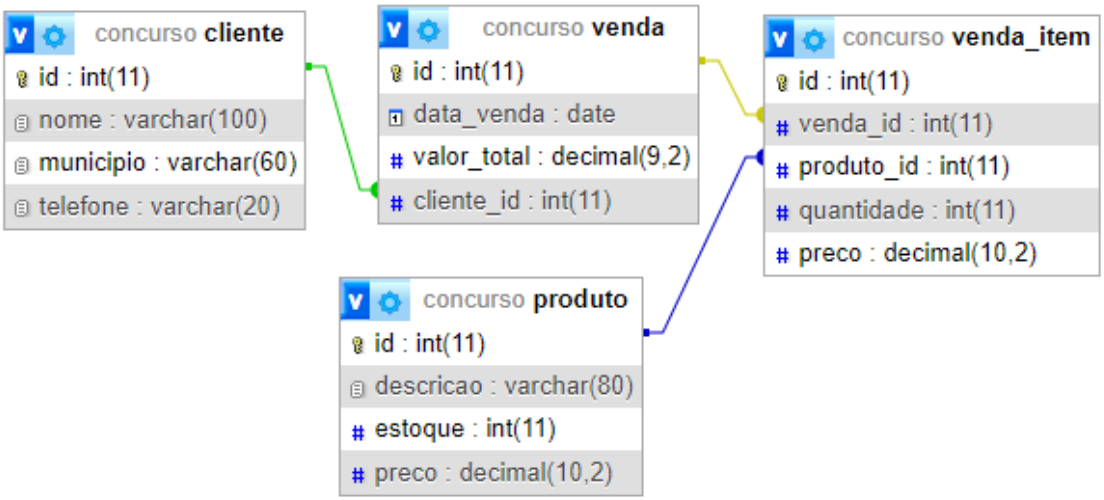
ii. O arquivo *"vendas.html"*, hospedado na raiz do servidor indicado anteriormente, que possua o código HTML abaixo:

```
<!DOCTYPE html><html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <title>Vendas</title>
   <link rel="stylesheet" href="estilo.css" type="text/css" />
   <script src="listagem-vendas.js" type="module" defer ></script>
</head>
<body>
   <h1>Vendas</h1>
   <div id="msg" >
      <!-- Exibir quaisquer mensagens aqui -->
   </div>
   <table>
        <thead>
          <tr>
            <th>Id</th>
            <th>Cliente</th>
            <th>Qtd. Vendas</th>
            <th>Valor Total (R$)</th>
          </tr>
       </thead>
       <tbody>
         <!-- Aqui a tabela vai ser renderizada -->
       </tbody>
       <tfoot>
         <!-- Aqui deve ser colocada a totalização -->
       </tfoot>
  </table>
</body>
</html>
```

iii. Que o arquivo *"estilo.css"*, utilizado por *"vendas.html"*, exista e que seu conteúdo seja irrelevante para os propósitos desta questão.

iv. O modelo de banco de dados abaixo, relativo a um banco de dados MySQL de nome *"concurso"*, que esteja localizado em *"localhost"*, executando na porta 3306, que utilize o conjunto de caracteres *(charset) "utf8"* e que possa ser acessado com o usuário *"desenvolvedor"* e a senha *"dev123456"*. Esse modelo representa que um cliente (um registro da tabela *“cliente”*) pode ter

muitas vendas (isso é, muitos registros da tabela *"venda"* associados). Cada venda (isso é, cada registro da tabela *"venda"*) pode ter muitos itens (ou seja, muitos registros da tabela *"venda_item"* associados) e cada um desses itens possui um produto (isso é, um registro da tabela *"produto"* associado).



v. O formato de objeto JSON descrito no código abaixo, que deverá ser gerado pelo lado servidor, via PHP, e processado pelo lado cliente, via JavaScript. Esse objeto contém os atributos *"error"* (do tipo *boolean*), *"msg"* (do tipo *string*) e *"data"* (do tipo *Array*). O atributo *"error"* indica se um erro qualquer ocorreu durante um processamento no servidor. O atributo *"msg"* pode conter uma mensagem para o usuário, seja ela de erro ou de sucesso. O atributo *"data"* pode conter um *array* de objetos (JSON) com dados quaisquer resultantes da contabilização de vendas por cliente. Cada objeto desse *array* de *"data"* contém os atributos *"id"* (*number*), *"cliente"* (*string*), *"quantidadeVendas"* (*number*) e *"valorTotal"* (*number*). Considere os dados abaixo como apenas um exemplo.

```
{ "error":false, "msg": "", "data": [
  { "id":11, "cliente": "Ana da Silva", "quantidadeVendas": 10, "valorTotal":
2985.92 },
  { "id":2, "cliente": "Beatriz Souza", "quantidadeVendas": 9, "valorTotal":
1999.35 }
] }
```

a) **(1,0 ponto)** Utilizando apenas JavaScript e a Fetch API, implemente o conteúdo do arquivo *"listagem-vendas.js"* (que é referenciado em *"vendas.html"*), para que seja capaz de executar as seguintes ações:

- Obter do servidor HTTP, do arquivo *"vendas.php"*, os dados de contabilização de vendas por cliente, para que esses sejam exibidos na tabela HTML. Esse arquivo PHP retorna um objeto no formato JSON explicado no item (v), acima.

- Adicionar linhas ao corpo da tabela HTML, em caso de sucesso no retorno do servidor (*"error"* contendo valor *false*), correspondentes aos dados das vendas por cliente recebidas do servidor (via atributo "*data*"). A tabela abaixo ilustra o preenchimento com possíveis dados do servidor.

Vendas

Id	Cliente	Qtd. Vendas	Valor Total (R$)
258	MARIA JOSÉ DUTRA	51	98678.92
103	PEDRO ANTONIO LOBO	4	29428.00
107	ANA MARIA DA SILVA	2	24641.17
521	PAULO ROBERTO TAVARES	7	8894.66
656	MARCIA FLORES	4	8477.80
686	ROBERTO MATOS	12	8192.04
98	MAURO QUINTANA	9	7343.85
7	OLAVO DA SILVA FREITAS	1	6555.34
97	JOSÉ DOS SANTOS	3	5407.50
661	PEDRO TORRES	4	4365.64
		97	195070.60

- Adicionar ao rodapé da tabela HTML uma linha com o somatório dos valores das colunas *"Qtd. Vendas"* e *"Valor Total (R$)"*. A distribuição das células dessa linha de rodapé deve ser exatamente igual à da tabela exemplo acima, com atenção para a primeira célula, que mescla as primeiras duas colunas.

- Exibir, em caso de quaisquer erros ou exceções, a mensagem correspondente no elemento indicado no HTML (*"div"*).

b) **(1,0 ponto)** Utilizando PHP e PDO, implemente, tratando quaisquer exceções lançadas, o conteúdo do arquivo *"vendas.php"* para que este:

- Estabeleça uma conexão com o banco de dados fornecido anteriormente, necessariamente através de uma função declarada em um arquivo *"conexao.php"*. Essa função deve retornar a instância de PDO realizada para estabelecer a conexão.

- Consulte desse banco de dados, apenas através de um único comando SQL, os dados de contabilização de vendas por cliente. Cada linha retornada pela consulta deve conter as seguintes colunas:
  - *"id"*, que deve corresponder ao código identificador do cliente e cujo valor deve ser único no resultado;
  - *"cliente"*, que deve corresponder ao nome do cliente;
  - *"quantidadeVendas",* que deve corresponder à quantidade total de vendas realizadas para o cliente;
  - *"valorTotal"*, correspondente ao valor total (R$) das vendas realizadas para o cliente, que deve ser obtido dos itens vendidos para o cliente, considerando o preço do item e a quantidade comprada. As linhas da tabela devem ser ordenadas por essa coluna, em ordem decrescente de valor.

- Responda a quaisquer requisições HTTP com um objeto JSON, cujo formato é indicado no item (v), acima. A resposta deve vir acompanhada do(s) cabeçalho(s) HTTP necessário(s). Em caso de sucesso, o atributo "*data*" deve ser preenchido com o resultado da consulta descrita acima, o atributo *"error"* deve ser preenchido com o valor *false* e o atributo *"msg"* deve ser retornado com uma *string* vazia (*""*). Em caso de qualquer erro ou exceção, o atributo *"error"* do objeto ser preenchido com *true*, o atributo *"msg"* deverá vir preenchido com a mensagem *"Erro ao consultar as vendas."* e o atributo *"data"* deverá vir vazio (*[ ]*). Exemplo para o caso de erro/exceção:

```
{
    "error":true,
    "msg": "Erro ao consultar as vendas.",
    “data”: [ ]
}
```

Questão 3) Considere um sistema desenvolvido em linguagem Java para uma revendedora de veículos automotores e acessórios. Observe atentamente todas as linhas de código do método *main()* desse sistema fictício no quadro abaixo.

```
06 public static void main(String[ ] args) {
07   //Aributos: modelo, ano, precoDeCusto
08   VeiculoDePasseio vp = new VeiculoDePasseio("Gol", 2021, 40000);
09   //Atributos: modelo, ano, precoDeCusto, capacidadeDeCargaEmKg
10   VeiculoUtilitario vu = new VeiculoUtilitario("Saveiro", 2019, 45000, 200);
11   List<VeiculoAutomotor> veiculos = new ArrayList<VeiculoAutomotor>();
12   veiculos.add(vp);
13   veiculos.add(vu);
14   //Atributos: modelo, ano, precoDeCusto, cilindradas
15   veiculos.add(new Motocicleta("XYBR", 2020, 28000, 250));
16   List<Tributavel> tributaveis = new ArrayList<Tributavel>();
17   tributaveis.add(vp);
18   tributaveis.add(vu);
19                                        //descricao, precoDeCusto
20   tributaveis.add(new Acessorio("Amortecedores", 2000));
21
22   System.out.println("Veículos automotores ordenados pelo preço de custo\n");
23   Collections.sort(veiculos);
24   for (VeiculoAutomotor v: veiculos)
25     System.out.println(v.getModelo() + " " + v.getAno() + " - P. custo: R$" +
                          v.getPrecoDeCusto() + " - P. venda: R$" + v.getPrecoDeVenda());
26
27   System.out.println("\nVeículos automotores ordenados pelo ano de fabricação\n");
28   Collections.sort(veiculos, new VeiculoAutomotorComparatorAno());
29   for (VeiculoAutomotor v: veiculos)
30     System.out.println(v.getModelo() + " " + v.getAno() + " - P. custo: R$" +
                          v.getPrecoDeCusto() + " - P. venda: R$" + v.getPrecoDeVenda());
31   System.out.println();
32   double totalEmValoresDeIpi = 0.0;
33   for (Tributavel t: tributaveis)
34     totalEmValoresDeIpi += t.getValorIpi();
35   System.out.println("Valor total em IPI: R$"+totalEmValoresDeIpi);
36 }
```

(2,0 pontos) Com base no método *main()* fornecido acima, escreva todo o código necessário para que o mesmo funcione corretamente. Para isso, utilize de forma adequada os preceitos da Programação Orientada a Objetos com muita atenção às informações fornecidas abaixo:

a) Só deve ser possível instanciar/criar objetos dos tipos *VeiculoDePasseio*, *VeiculoUtilitario*, *Motocicleta*, *Acessorio* e *VeiculoAutomotorComparatorAno* em memória.

b) Não é necessário declarar nenhum *import* em nenhuma parte do código.

c) Não é necessário escrever métodos *get* e *set*. Partiremos da premissa de que os métodos já estarão implementados.

d) Só deve ser possível instanciar *VeiculoDePasseio*, *VeiculoUtilitario* e *Motocicleta* fornecendo valores para os atributos *modelo*, *ano* e *precoDeCusto*. Essa regra é comum e, portanto, deve estar implementada em um único lugar.

e) Para instanciar um *VeiculoUtilitario*, é obrigatório fornecer ainda um valor para o atributo *capacidadeDeCargaEmKg* (linhas 9 e 10).

f) Para instanciar uma *Motocicleta*, é obrigatório fornecer ainda um valor para o atributo *cilindradas* (linhas 14 e 15).

g) Só deve ser possível instanciar *Acessorio* fornecendo valores para os atributos *descricao* e *precoDeCusto* (linhas 19 e 20).

h) Como pode ser observado nas linhas 25 e 30, todos os objetos adicionados à lista denominada *veiculos* sabem fornecer seu preço de venda. No entanto, as regras são diferentes para cada objeto. O preço de venda de um *VeiculoDePasseio* é obtido aplicando um percentual de 50% sobre seu preço de custo (atributo *precoDeCusto*). O preço de venda de um *VeiculoUtilitario* é obtido aplicando um percentual de 45% sobre seu preço de custo. O preço de venda de uma *Motocicleta* é obtido aplicando um percentual de 40% sobre seu preço de custo.

i) Observe as linhas 22 e 23 e cuide para que seja possível ordenar a lista denominada *veiculos* pelo atributo *precoDeCusto*.

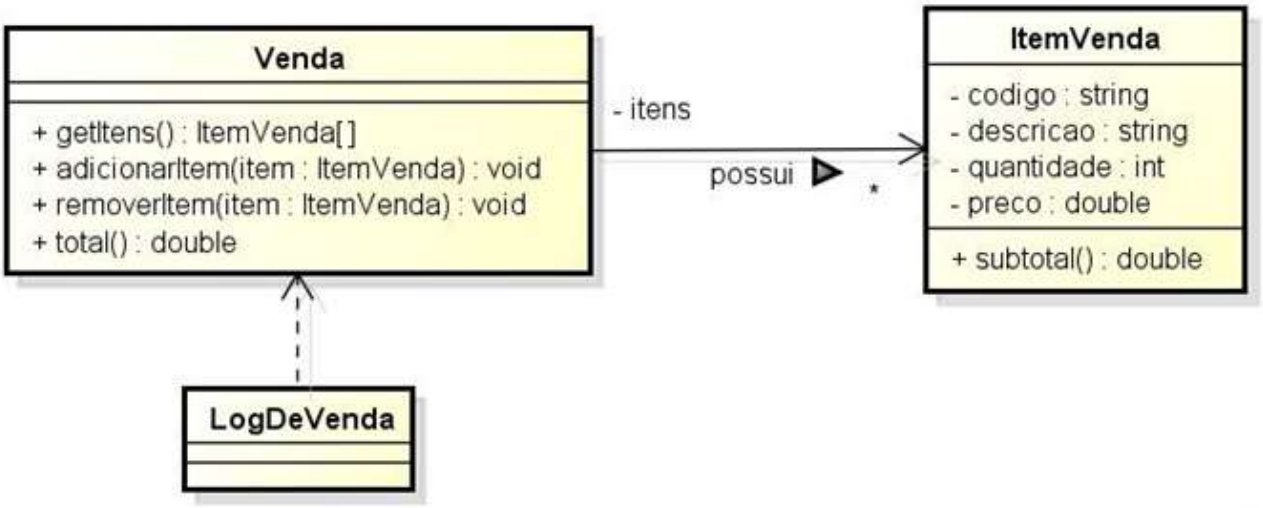
j) Observe as linhas 27 e 28 e cuide para que seja possível ordenar a lista denominada *veiculos* alternativamente pelo ano.

k) Somente os objetos adicionados à lista denominada *tributaveis* devem saber fornecer seu valor de IPI (linhas 33 e 34). O IPI sobre objetos do tipo *Acessorio* é de 18% sobre seu preço de custo (atributo *precoDeCusto*) e o IPI sobre objetos dos tipos *VeiculoUtilitario* e *VeiculoDePasseio* é de 30% sobre seu preço de custo.

l) Não deve haver duplicação desnecessária de regras no código. Existe um lugar específico para a implementação de cada método de modo a não causar impactos indesejados em outras partes do sistema, como prejuízos para o seu funcionamento ou para sua manutenção.

Questão 4) Utilize o Padrão de Projeto GoF mais adequado para solucionar cada problema (subquestão) a seguir, realizando a implementação na linguagem Java ou, alternativamente, na linguagem PHP com tipagem de dados. Para cada caso, identifique o nome do Padrão de Projeto no topo da solução, implemente as classes apresentadas realizando as modificações necessárias e crie (implemente) quaisquer outras classes ou interfaces que forem necessárias para utilizar o Padrão de Projeto.

a) **(1,0 ponto)** Considere como base, o Diagrama de Classes (UML) a seguir:



A classe *Venda* representa uma venda. Seu método *getItens()* retorna seus itens, que são objetos da classe *ItemVenda*. Seu método *total()* retorna o total da venda, que é o somatório dos subtotais de seus itens. A classe *ItemVenda* representa um item da venda e seu método *subtotal()* retorna a multiplicação de seu preço por sua quantidade.

Toda vez que o total da venda mudar, é preciso que uma instância qualquer da classe *LogDeVenda*, que tenha sido inscrita para monitorar *Venda*, seja notificada a respeito. Isso deve ocorrer sem que *Venda* tenha conhecimento (acoplamento) direto da classe *LogDeVenda* e sem intermediários. Essa mudança no total da venda somente ocorre quando um item for adicionado ou removido da mesma, através dos métodos *adicionarItem()* e *removerItem()*. Uma *Venda* deve notificar todos os objetos inscritos para tal.

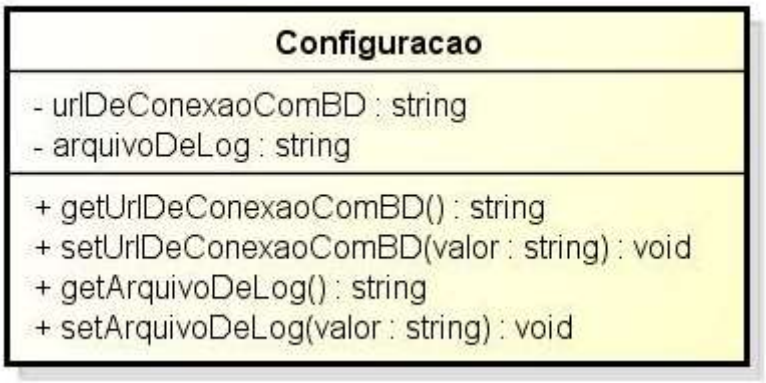
Quando notificada, a instância de *LogDeVenda* deve imprimir no console a frase *"O total da venda mudou de R$ A para R$ B"*, em que *"A"* é o valor total antigo da venda e *"B"* é o valor total novo da venda.

Observações:

- Não é preciso declarar o pacote de classes ou interfaces implementadas.
- Não é preciso realizar importações de declarações.

- Não é preciso implementar a classe *ItemVenda*. As demais classes da solução podem considerar que essa classe exista, tenha métodos *getters* e *setters* e o método *subtotal()*.
- Não é preciso implementar os métodos *getItens()* e *total()*, da classe *Venda*. Pode-se considerar que os mesmos existam e realizem os comportamentos descritos anteriormente.
- Não é preciso arredondar ou truncar o total da venda (por exemplo, para duas casas decimais) visando a impressão de seu valor.

b) **(1,0 ponto)** Considere como base o Diagrama de Classes (UML) a seguir:



A classe *Configuracao* representa a configuração de um sistema e deve garantir a criação de somente uma (única) instância da mesma, que poderá ser utilizada em qualquer parte do sistema.

Observações:

- Não é preciso declarar o pacote de classes ou interfaces implementadas.
- Não é preciso realizar importações de declarações.

Questão 5) A respeito do Planejamento e Desempenho de Custos, quatro medidas são de grande importância:

- Custo Orçado Total (COT)
- Custo Orçado Acumulado (COC)
- Custo Real Acumulado (CRC)
- Valor Agregado Acumulado (VAC)

Determinado projeto foi dividido em quatro pacotes de trabalho. O Custo Orçado Total (COT) dos diversos elementos necessários para o seu desenvolvimento, como mão de obra, materiais, fornecedores, consultores, aluguéis de equipamentos ou instalações e viagens, foi estimado e fracionado ao longo das unidades de tempo (em semanas) previstas para cada um desses pacotes, de acordo com a tabela 1.

Tabela 1:

Pacote	**COT**	**Semana**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**15**	5	7.5	2.5							
2	**42**		6	6	9	9	6	6			
3	**28**					5	5	13	5		
4	**15**								2.5	2.5	10
Total	**100**	**5**	**13.5**	**8.5**	**9**	**14**	**11**	**19**	**7.5**	**2.5**	**10**

Quantias expressas em milhares de reais.

Sobre o Custo Orçado Acumulado (COC) ao longo de cada semana planejada, esse pode ser calculado acrescentando uma linha na tabela acima. Nessa linha, para a primeira semana, o COC e o COT semanal se equivalem. Quanto às demais semanas, deve-se somar, para cada uma delas, o COT dessa semana ao COC da semana imediatamente anterior. A tabela 2 expressa essa situação:

Tabela 2:

Pacote	**COT**	**Semana**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**15**	5	7.5	2.5							
2	**42**		6	6	9	9	6	6			
3	**28**					5	5	13	5		
4	**15**								2.5	2.5	10
Total	**100**	**5**	**13.5**	**8.5**	**9**	**14**	**11**	**19**	**7.5**	**2.5**	**10**
COC		**5**	**18.5**	**27**	**36**	**50**	**61**	**80**	**87.5**	**90**	**100**

Quantias expressas em milhares de reais.

A tabela 3 ilustra o Custo Real por Período para o projeto, tendo sido passadas 5 semanas desde seu início.

Tabela 3:

Pacote	Semana				
	1	2	3	4	5
1	5	8	3		
2		6	5.5	9.5	9
3					4
4					

Quantias expressas em milhares de reais.

a) **(0,2 ponto)** Qual é o Custo Real Acumulado (CRC) por período desde a semana 1 até a semana 5?

b) **(0,4 ponto)** A fim de se estudar as metas de um projeto, previstas e atingidas em determinado momento, a que conclusão se pode chegar a respeito do planejamento e do controle desse mesmo projeto com base nas informações contidas no COT, COC e CRC? Explique.

c) **(0,6 ponto)** Escreva o conceito de Valor Agregado Acumulado (VAC) no que se refere à gerência de projetos e explique como ele é calculado e de que forma essa medida é capaz de indicar quando surge a necessidade de se tomar ações que visem o controle de custos.

d) **(0,8 ponto)** Sendo conhecidos os tipos de registro descritos abaixo em linguagem C,

```
typedef struct _tabela_
{
  float ** valores;
  int linhas;
  int colunas;
} tabela;
```

```
typedef struct _lista_
{
  float * valores;
  int colunas;
} lista;
```

use-os para escrever a função *calculaValorRealAcumulado*, também em linguagem C, chamada por uma linha de comando presente no trecho de programa abaixo:

```
tabela custoRealPorPeriodo;
lista custoRealAcumuladoPorPeriodo;
/*... */
calculaValorRealAcumulado(custoRealPorPeriodo, &custoRealAcumuladoPorPeriodo);
```

Essa função, portanto, deve receber dois parâmetros. O primeiro parâmetro é uma tabela que armazena o custo real de cada pacote que foi gasto no decorrer de cada semana. Os pacotes constituem as linhas da tabela e as semanas constituem as colunas da tabela, tal como na tabela 3. O segundo parâmetro refere-se a uma lista que precisa ter seus valores calculados. Essa lista deve conter o custo real acumulado do projeto ao longo das semanas.