1)	**3,5**	
2)	**3,0**	
3)	**3,5**	

Fundação CECIERJ – Vice Presidência de Educação Superior à Distância
Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação de Aplicações Web
Professores: Diego Passos e Uéverton dos Santos Souza
AD1 – 2º Semestre de 2018

Estão sendo avaliados o uso das estruturas básicas de repetição e condição, a criação e uso de funções, manipulação de vetores e programação orientada a objetos na linguagem PHP.

1) Em diversos sistemas computacionais, a informação manipulada pode ser corrompida de alguma forma. Por exemplo, devido a defeitos em componentes ou variações na alimentação, um bit de um dado armazenado na memória RAM pode ter seu valor alterado de 0 para 1 ou vice-versa. Para dar mais confiabilidade a esse tipo de sistema, é comum a utilização de *códigos de correção de erros*. Esses códigos introduzem bits redundantes a uma certa informação original de forma que, sob certas condições, eles possam recuperar o valor correto do dado em caso de corrupção.

Um código particularmente popular é o chamado Hamming(7,4). Este método trabalha com blocos de 4 bits de dados. Para cada bloco, o método adiciona mais 3 bits redundantes, representando a informação original com um total de 4 + 3 = 7 bits. Cada bit redundante é calculado de acordo com o valor de três dos quatro bits de dados, sendo:

- **Primeiro bit de redundância**: computado sobre os valores do **primeiro, segundo e quarto bits de dados**.
- **Segundo bit de redundância**: computado sobre os valores do **primeiro, terceiro e quarto bits de dados**.

- **Terceiro bit de redundância**: computado sobre os valores do **segundo, terceiro e quarto bits de dados**.

Mais especificamente, o valor de um bit de redundância é 1 se uma das seguintes condições for verdade:

- **Todos** os bits de dados correspondentes são iguais a 1; **ou**
- Se **exatamente um** dos bits de dados correspondentes é igual a 1.

Por exemplo, para o bloco `0111`, os bits de redundância gerados seriam:

- Primeiro bit de redundância é 0, porque há dois bits com valor 1 entre o primeiro, segundo e quarto bits de dados.
- Segundo bit de redundância é 0, porque há dois bits com valor 1 entre o primeiro, terceiro e quarto bits de dados.
- Terceiro bit de redundância é 1, porque o segundo, o terceiro e o quarto bits de dados são todos iguais a 1.

Com base nessa descrição, pede-se:

a) (**1,5 pontos**) Escreva uma função em PHP que receba 4 parâmetros inteiros representando os valores dos bits de dados de um bloco, **calcule e imprima os bits de redundância** de acordo com a codificação Hamming (7,4). Sua função **não precisa** verificar a validade dos parâmetros recebidos.

b) (**1,0 ponto**) Escreva uma função em PHP que receba uma string **formada apenas por caracteres '0' e '1' cujo comprimento é múltiplo de 4 (tendo ao menos quatro caracteres)** representando uma sequência de bits qualquer. Sua função deverá **quebrar essa sequência em blocos de 4 bits** e, para cada bloco, deverá **chamar a função** implementada no item a) com os parâmetros adequados. Novamente, **você pode assumir que a string recebida como parâmetro dessa função é correta**, sem precisar verificar o formato.

c) (**1,0 ponto**) Escreva **uma expressão regular** que possa ser usada para validar o formato da string passada como parâmetro para a função implementada para o item b). Em particular, sua expressão regular deverá casar apenas com **strings formadas somente pelos caracteres '0' e '1' cujo comprimento é múltiplo de 4 (tendo ao menos quatro caracteres)**.

2) Suponha que a empresa na qual você trabalha tenha sido contratada para desenvolver um *software* para realizar análises estatísticas nos jogos da Copa do Mundo. Em particular, você recebe a tarefa de realizar uma revisão do código que já foi escrito. Em certo momento, você se depara com a seguinte função:

```
1: function a($MinutoDosGols) {
2:
3:       $MaximoDeGols = 0;
4:       $MinutoDoMaximo = -1;
5:
6:       for ($i = 0; $i < 90; $i++) {
7:
8:            $GolsNoMinuto = 0;
9:            for ($j = 0; $j < count($MinutoDosGols); $j++) {
10:
11:                 if ($MinutoDosGols[$j] == $i) {
12:
13:                       $GolsNoMinuto = $GolsNoMinuto + 1;
14:                 }
15:           }
16:
17:           if ($GolsNoMinuto > $MaximoDeGols) {
18:
19:                 $MaximoDeGols = $GolsNoMinuto;
20:                 $MinutoDoMaximo = $i;
21:           }
22:      }
23:
24:      echo $MinutoDoMaximo . "\n";
25:      echo $MaximoDeGols . "\n";
26: }
```

Sabendo que o parâmetro `$MinutoDosGols` é um vetor em que cada posição armazena o minuto do jogo (de 0 a 90) em que cada gol marcado na copa ocorreu, pede-se:

a) (**1,0 ponto**) Descreva em uma frase o **objetivo da função** (note que você deve descrever o que a função faz como um todo, e não ler o que está sendo feito a cada linha/trecho do código).

b) (**2,0 pontos**) Repare que o código realiza duas repetições aninhadas (linhas 6 e 9). Reescreva a função **de forma que ela realize exatamente o mesmo objetivo, mas sem o uso de repetições aninhadas (**sua função ainda poderá conter uma ou mais repetições, mas elas não devem ser aninhadas).

3) Suponha que você tenha sido contratado para escrever um sistema simples de controle de folha de pagamento de uma empresa. Você decide desenvolvê-lo em PHP, usando orientação a objetos. Pede-se:

a) (**Valor: 2,5 pontos**) Escreva uma pequena classe chamada `Funcionario` que representa um funcionário da empresa. Para efeito dessa questão, considere que cada funcionário possui um nome, um CPF, uma data de nascimento e um salário. Sua classe deverá possuir métodos que permitam consultar e alterar cada uma dessas propriedades.

b) (**Valor: 1,0 ponto**) Escreva um trecho de código em PHP que crie uma instância da sua classe `Funcionario` com os seguintes dados:

- Nome: José Silva
- CPF: 012.345.678-90
- Data de Nascimento: 20/07/1967
- Salário: R$ 2000,00.