UNIVERSIDADE ABERTA
www.uab.pt

Ministério da Educação e Ciência

p-Fólio

U.C. 21025

Desenvolvimento de Software

15 de junho de 2015

INSTRUÇÕES

PARA A RESOLUÇÃO DO EXAME, ACONSELHA-SE QUE LEIA ATENTAMENTE O SEGUINTE:

1. Para a resolução deste **p – Fólio** aconselha-se que:
2. Verifique o exemplar que lhe foi entregue e, no caso de estar incompleto ou com qualquer deficiência, dirija-se ao professor vigilante.
3. O **p-fólio** é composto por 2 partes.
4. O teste é sem consulta e termina com a palavra **FIM**.
5. Utilize, sempre, uma letra legível e não use uma caneta de outra cor que não seja o preto ou o azul - as respostas a lápis não serão consideradas.
6. Tenha em atenção que o **p-Fólio** tem a duração de 1 hora e 30 minutos.

Bom Trabalho!

Nome: .....

N.º de Estudante: ..... B. I. n.º .....

Turma: ..... Assinatura do Vigilante: .....

1.ª Parte (8 Valores)

As alíneas seguintes são todas de resposta curta.

1. No desenvolvimento de uma aplicação móvel de apoio a jogadores de snooker (sugestão de tacadas), ocorreram vários problemas. Identifique os problemas seguintes como defeitos (fault), enganos (mistake), erros (error) ou falhas (failure).

a) A aplicação sugere uma tacada que requer uma tabela num dos buracos da mesa. ~~engano~~ defeito

b) A variável com a série do jogador não é reinicializada após o final de um jogo. ~~defeito~~ engano

2. Que membros da equipa de desenvolvimento de software devem participar nas seguintes tarefas?

a) Debater com a empresa distribuidora da aplicação, para definir as funcionalidades a fornecer ao utilizador final: Analista

b) Corrigir um bug na aplicação, que a impede de identificar corretamente as dimensões de algumas mesas de snooker programador + tester?

c) Teste das várias unidades da aplicação tester

3. Classifique os seguintes requisitos como sendo funcionais ou não funcionais.

a) O sistema deve consultar um serviço Web para obter publicidade a apresentar durante as sugestões de jogadas. não funcional?

b) O sistema deve funcionar em sistemas Android e iOS. não funcional

4. Identifique o tipo de dependência entre componentes de software nas seguintes situações:

a) Quando apenas dados são passados entre componentes lógica?

b) Quando um componente é controlado pelos argumentos invocados de outro sequencial?

5. Identifique os níveis de coesão entre componentes de software nas seguintes situações:

a) Quando as partes têm de produzir ou alterar o mesmo ficheiro. comunicacional?

b) Quando as partes estão juntas pela altura em que são chamadas, por exemplo, todos os procedimentos de encerramento da aplicação. temporal

Nome: .....

N.º de Estudante: ..... B. I. n.º .....

Turma: ..... Assinatura do Vigilante: .....

6. **Que tipo de testes devem ser realizados para detetar os seguintes defeitos:**

   a) Um bloco de código foi copiado e alterado para reagir a situações similares, mas numa delas o programador esqueceu-se de o alterar, mantendo-o exatamente igual. integração

   b) Há um defeito no algoritmo utilizado, pelo que se o programa for testado com algumas entradas específicas, os resultados (saídas) correspondentes não estarão conformes à especificação. ~~estrutural~~ funcional

7. **Um gestor de testes preocupado com a fiabilidade do sistema decidiu introduzir defeitos no código para estimar os defeitos em falta. Suponha que foram introduzidos 50 defeitos no código. A equipa de testes encontrou 60 defeitos dos quais 30 correspondiam a defeitos introduzidos pelo gestor de testes. Qual a estimativa do número total de defeitos no código? Quantos defeitos ainda faltam encontrar?**

8. **Diga que tipo de manutenção (preventiva, adaptativa, aperfeiçoativa ou corretiva) deve ocorrer nas seguintes situações (pode não ser necessária qualquer ação de manutenção):**

   a) Num momento de inspiração, o produtor da aplicação de apoio a jogadores de snooker apercebe-se de que pode usar a deteção de reflexos de luz na bola verde para melhorar a capacidade da aplicação para a distinguir do verde do pano da mesa. Esta situação deve conduzir a uma manutenção aperfeiçoativa, porque consiste no aperfeiçoamento da aplicação?

   b) O cliente decide que a aplicação, embora já operacional em plataformas Android, deve ser disponibilizada nos óculos sucessores dos Google Glass, com características de ecrã muito diferentes dos *smartphones* tradicionais. Esta situação deve conduzir a uma manutenção adaptativa, porque pretende-se adaptar a aplicação a outro tipo de ecrã que não o dos smartphones?

2.ª Parte (4 Valores)

Atendendo às normas de escrita de código em anexo (igual às disponibilizadas no espaço da unidade curricular), escreva uma função de cálculo do ângulo de deflexão de uma bola de bilhar, que dado um ângulo de impacte entre 0° e 90° devolve o seu suplementar (diferença para 180°). Caso o ângulo não esteja entre 0° e 90° mas sim entre 90° e 180°, a função devolve o negativo do suplementar. Caso o ângulo não esteja entre 0° e 180°, a função devolve -1. (Ex: Deflexao(60): 120; Deflexao(120): -60; Deflexao(90): 90; Deflexao(270): -1.)

Nome: .....

N.º de Estudante: ..... B. I. n.º .....

Turma: ..... Assinatura do Vigilante: .....

Anexo - Norma de Escrita de Código

1. Formatação

1.1 Indentação: Cada linha de código deve começar por 4 espaços por cada bloco de código em que estiver inserido. É aceitável em vez de 4 espaços utilizar outro valor, desde que seja sempre o mesmo para todo o ficheiro.

1.2 Tamanho das linhas: As linhas tanto código como de comentários não devem exceder os 80 caracteres. Caso tal não seja possível, a linha de código deve ocupar mais que uma linha. Conta para efeitos de indentação a segunda linha e seguintes, como se estivessem dentro de mais um bloco de código. Uma linha de comentário que continue na linha seguinte mantém a indentação.

1.3 Argumentos: As listas de argumentos, tanto na definição como na utilização, não são considerados blocos de código. Assim devem ser escritos na mesma linha, se possível, caso contrário aplica-se o ponto 1.2.

1.4 Início e fim de bloco: A marca de início e de fim de bloco de código deve utilizar uma linha para o início e outra para o fim do bloco. As linguagens funcionais podem ignorar esta regra, uma vez que em geral o primeiro elemento do bloco é um identificador e convém que fique na mesma linha que a abertura de bloco.

2. Comentários

2.1 Localização: Os comentários devem anteceder o código que comentam. Pode-se colocar pequenos comentários ao bloco na própria linha de início ou fim de bloco de código.

2.2 Comentário de classe: No início de cada classe deve existir um comentário de classe. Neste comentário deve conter informação sumária sobre a classe. Dá-se liberdade de incluir mais ou menos campos conforme apropriado, no entanto o comentário de classe deve ter obrigatoriamente os campos (inglês/português):

- Author/Autor - uma ou mais pessoas que contribuíram para a classe
- Last change/Última modificação - data da última modificação
- Description/Descrição - descrição da estrutura de dados e breve descrição dos algoritmos envolvidos, esclarecendo o que os algoritmos fazem e não como fazem

2.3 Comentário de método: Cada método deve ter um comentário inicial, idêntico ao comentário de classe, acrescentando:

- Descrição do algoritmo, não só o que faz mas também como o faz;
- A complexidade temporal do algoritmo, sempre que possível. No início dos blocos de código pode-se colocar também a complexidade temporal do bloco de código;
- Pré e pós-condições de utilização do método, por exemplo, os elementos de uma lista estarem ordenados;
- Incluir eventuais restrições de utilização, por exemplo, o valor máximo para cada argumento do método para o qual o método foi testado e funciona bem. Após serem realizados testes, os resultados dos testes devem ser também incluídos.

2.4 Comentário de linha: Devem ser colocados comentários de linha apenas em zonas complexas, de forma a complementar a descrição do algoritmo feita no comentário de método. Os comentários de linha devem assumir que o leitor conhece a linguagem de programação, pelo que não devem repetir o que está no código.

3. Nomes

3.1 Variáveis iteradoras inteiras: Os nomes das variáveis devem começar por letras minúsculas. As variáveis iteradoras inteiras devem ser chamadas i, j e k. Devem ser reutilizadas, devendo o k ser utilizado apenas se o i e j estiverem em utilização. Caso sejam necessárias mais que três variáveis iteradoras, adicionar um número à letra i (i1, i2, ...). Caso as variáveis iteradoras sejam reais ou de outra natureza, utilizar as letras l, m e n.

3.2 Variáveis não iteradoras: Para as variáveis não iteradoras deve-se utilizar nomes elucidativos. Na junção de dois nomes, a letra inicial do segundo nome deve ser maiúscula. A primeira letra deve ser sempre minúscula. Caso não existam nomes elucidativos, utilizar as letras u, v e w para os valores inteiros, x, y e z para os valores reais, a, b e c para nomes (strings), e p, q e r para apontadores ou variáveis de outra natureza.

3.3 Constantes e macros: Constantes ou macros devem ser escritos em maiúsculas, e as palavras separadas por um "underscore": '_'.

Nome: .....

N.º de Estudante: ..... B. I. n.º .....

Turma: ..... Assinatura do Vigilante: .....

3.4 Estruturas e classes: Os nomes das estruturas ou classes têm de ter sempre nomes elucidativos e devem começar por T se é um tipo abstracto de dados, S se for uma estrutura, e C se for uma classe no geral, D se for uma classe para caixas de diálogo de uma interface gráfica. Na junção de dois nomes, a letra inicial de cada nome deve ser maiúscula.

3.5 Métodos: Os nomes dos métodos devem ter sempre nomes elucidativos. No caso de programação não orientada por objectos, o nome das funções e procedimentos deve ser antecedida do nome ou iniciais do módulo a que pertencem. Na junção de dois nomes, a letra inicial de cada nome deve ser maiúscula. Não utilizar o mesmo nome para funções ou métodos com argumentos diferentes, a não ser que as funções ou métodos sejam apenas diferentes versões. Em princípio, o nome de um procedimento deve conter um verbo no infinitivo, enquanto que o nome de uma função deverá ter o nome da grandeza que calcula, não necessitando de verbo. As funções com resultado booleano devem conter um verbo no presente ou adjectivo de forma a serem utilizadas dentro de condicionais.

4. Estilo

4.1 Blocos de código: Não abrir um bloco de código para um só comando, excepto para evitar confusão por exemplo numa cadeia if/else.

4.2 Repetições: Não repetir uma sequência de comandos. Muitas vezes por facilidade ao escrever código, repetem-se comandos quando poderiam ser postos dentro de um ciclo.

4.3 Tamanho dos métodos: Não fazer métodos demasiado grandes (mais de 100 linhas de código). Nesse caso agrupar comandos relacionados em métodos, mesmo que o método venha a ser chamado uma só vez.

4.4 Comandos numa linha: Iniciar sempre uma nova linha para um comando, ou seja, não colocar antes de um comando um condicional, ciclo, ou outro comando.

FIM