CONCURSO PD – 2ª Festa da Programação

Regras da Segunda Festa da Programação

Deverão ter em conta as seguintes regras, as quais serão tidas em conta na validação dos programas:

- Só podem utilizar um e apenas um return em cada função, exceto se se tratar de uma função de busca, do estilo da função **ints_find**, que usa o esquema for-if-return-return, cf. aula 14.
- Todos os programas têm obrigatoriamente de ter uma função de teste.
- Todas as funções devolvem um valor inteiro, double ou string, exceto a função de teste.
- **Não podem usar variáveis nem arrays globais**, apenas podem definir constantes que podem ser utilizadas globalmente.
- Utilize o esquema de main() com as diversas opções dos problemas que é utilizado nos últimos concursos.
- Deverá utilizar **testes unitários** com pelo menos 3 exemplos nos programas B e C.
- **Programas aceites que não cumpram estas regras serão penalizados.**

Programa A – Mundial de Futebol

FIFA WORLD CUP Qatar2022

Portugal estava no mundial no Catar 😊. A Federação pretende efetuar uma análise geral do esforço dos jogadores através dos quilómetros (km) percorridos e das suas biometrias, peso inicial antes do jogo e peso final após o jogo. Sabemos que Portugal jogaria 3 jogos da fase de grupos, não sabemos quantos mais jogaria, pelo que apenas contarão estes 3 jogos para a análise. Registar-se-ão os dados dos 10 jogadores de campo, excluindo o guarda-redes. As substituições não interessam nestas contas pois serão agregadas e registadas nas entradas dos substituídos, já que o objetivo é a análise do esforço coletivo.

A Federação quer saber, para toda a equipa e no conjunto dos 3 jogos, a média, a moda e a mediana dos km percorridos de todos os jogadores. Pretende ainda saber, o total de peso perdido geral de todos os jogadores no conjunto dos 3 jogos.

Nota: A moda é o número mais repetido num array. Ex. array = 1,2,2,3,4,5,6 a moda é 2. A mediana calcula-se, num array cujo total de elementos é ímpar, após ordenação crescente, escolhendo o elemento do meio. Num array cujo número total de elementos é par, calcula-se através da divisão dos dois valores do meio após uma ordenação crescente do array. Em todos os testes, o tamanho dos arrays é um número par, garantidamente. Ex: array de comprimento par ordenado = 1,2,3,4,5,6 a mediana será $\frac{3+4}{2} = 3.5$.

Tarefa: Utilizando um único array de entrada de **doubles** e para o conjunto dos 3 jogos, escreva a função **avg_km** que devolve a média de km percorridos. Escreva uma segunda função, **mode_km,** que devolva a moda dos km percorridos e numa terceira função, **med_km**, ordene e devolva a mediana dos km percorridos. Escreva ainda uma quarta função, **total_weight,** que devolve o total de peso perdido. Finalmente, escreva a função **test_worldcup** como função de teste.

Input: Na tutoria está um ficheiro pA.txt que podem utilizar por redireccionamento.

6.2 72.3 70.2

5.4 82.2 80.0

5.5 79.8 77.5

… até ao final dos 10 jogadores por jogo x 3 jogos.

Explicação:

6.2 são os km percorridos, 72.3 os kg iniciais e 70.2 os kg finais do primeiro jogador no primeiro jogo e sucessivamente para os restantes 10 jogadores e 3 jogos.

Output:

6.04 5.70 5.95 44.60\n.

Os valores são calculados com todo o array e na totalidade dos 3 jogos.

O primeiro valor (6.04) é a media geral de km percorridos, o segundo valor (5.70) é a moda geral de km percorridos, o terceiro valor (5.95) é a mediana geral de km percorridos, o quarto valor (44.60) é o total de peso perdido por todos os jogadores nos 3 jogos.

Note bem: todos os números double no output devem ser escritos com duas casas decimais.

Submeta no Problema A.

Programa B – Combustível

Os combustíveis estão caríssimos e eu tenho a mania da poupança!

Assim, resolvi fazer uma lista de gasolineiras com a distância destas a minha casa e quais os preços que praticam no gasóleo simples, combustível que utilizo no meu automóvel.

Para me ajudar, decidir fazer um programa que determine qual a gasolineira que mais me compensa, mediante o valor que pretendo abastecer, a distância e o consumo do meu carro de **6.2 Litros/100**.

Tarefa:

Escreva na função **test_fuel** que fará a execução da leitura de dois arrays, através da função **doubles_get_until** da biblioteca **our_doubles.h,** utilizando o terminador **-1**.

No primeiro array, ficam registadas as distâncias e no segundo, o valor praticado do gasóleo na gasolineira, nos índices correspondentes.

Escreva uma função **gas** que calcule o **índice** da gasolineira que permite abastecer mais litros de gasóleo, mediante o valor que pretendo abastecer, a distância que se encontra e o consumo do meu automóvel. Escreva uma segunda função **gas_liters,** que calcule o número de litros que, mediante o valor dado, consigo abastecer, descontando o consumo feito na deslocação desde a minha casa até à gasolineira e volta.

Não se esqueça de compilar juntamente com os ficheiros **our_doubles.c** e o **our_ints.c**, estilo **gcc -Wall my_party.c our_doubles.c our_ints.c**.

Input:

7.2 15.4 16.3 22.5 17.9 35.23 2.4

1.629 1.573 1.654 1.618 1.622 1.476 1.712

Sessão interativa:

50

1 29.9

10

6 5.5

25

0 14.5

Note bem: os números double devem ser escritos com uma casa decimal.

Explicação:

Entrados 50€, a função calculou que a gasolineira que mais compensa é a do índice 1. Os 50€ deveriam dar para atestar 31.8 Litros, no entanto, como tenho de retirar o número de litros gastos na deslocação, restam apenas 29.9 Litros. Para 10€ euros compensa a gasolineira com o índice 6, conseguindo atestar 5.5 Litros e para 25€ compensa a gasolineira do índice 0, atestando 14.5 Litros com o valor entrado e sem o consumo da deslocação.

Submeta no problema B.

Programa C – Passwords

Ligaram-me do meu banco para avisar que houve um ataque informático e que a minha password ficou comprometida. Disseram-me que, por segurança, os meus acessos ficam bloqueados até um criar uma password nova.

Felizmente, tenho um esquema secreto que me permite gerar passwords sempre novas sem correr o risco de as esquecer.

Baseia-se nas letras de algumas canções dos Pink Floyd que sei de cor: de cada vez que preciso de uma password nova escolho um verso e "processo-o" mentalmente para criar a password. É simples:

Primeiro construo uma cadeia de caracteres com a primeira letra de cada palavra do verso. Por exemplo, se o verso for:

```
All in all it's just another brick in the wall
```

a cadeia será:

```
Aiaijabitw
```

Depois, para ficar mais complicado, coloco entre cada duas letras o algarismo das unidades da posição do espaço que separa as palavras das quais aquelas letras são as letras iniciais. Fica assim:

```
A3i6a0i5j0a8b4i7t1w
```

Repare: entre as duas primeiras palavras — "All" e "in" — há um espaço na posição de índice 3. Logo surge um algarismo '3' entre o 'A' e o 'i'. A seguir há espaços nas posições 6, 10, 15, 20, 28, 34, 37, 41. Por isso, os algarismos entre as letras, depois do 3, são 6, 0, 5, 0, 8, 4, 7 e 1.

Tarefa:

Escreva um programa que leia interactivamente da consola uma sequência de linhas onde em cada linha há uma sequência de palavras e para cada linha escreva na consola a password gerada segundo as regras indicadas.

Input:

Em cada linha do input, as palavras da sequência vêm separadas por caracteres de sublinhado, '_' (e não por espaços). Assim, do ponto de vista técnico, cada linha pode ser lida de uma vez com um scanf.

Não haverá linhas com mais que 80 caracteres. Não haverá linhas vazias. Não haverá caracteres de sublinhado seguidos. Não haverá caracteres espaço. O primeiro caractere da linha não será um sublinhado e o último também não.

Output:

Por cada linha lida, o programa responde imediatamente, escrevendo uma linha com a password calculada.

Exemplo de sessão iterativa:

```
agua_mole_em_pedra_dura
a4m9e2p8d
aaa_bbbbb
a3b
Teachers_leave_them_kids_alone
T8l4t9k4a
Money
M
All_in_all_it's_just_another_brick_in_the_wall
A3i6a0i5j0a8b4i7t1w
```

A primeira linha foi escrita pelo utilizador. A segunda contém a resposta do programa. E assim sucessivamente.