Nome:	Número:	Assinatura:

Universidade do Algarve

Programação Imperativa

Terceira Festa de Programação Imperativa

Aviso Geral

- Esta prova é uma prova "papel e caneta", sem consulta. **Só é autorizado ter na mesa da prova caneta, folhas de teste e folhas de rascunho virgens.**

  Qualquer outro material ou recurso **não é autorizado** durante a festa e **qualifica-se como fraude**.

- Qualquer uso de material indevido (telemóveis, *chats*, pdfs etc...) é sancionado com **reprovação imediata à UC de Programação Imperativa e será assinalada às autoridades académicas competentes**.

- Qualquer comportamento indevido, não autorizado ou fraude académica, etc... **é sancionado com reprovação imediata à UC de Programação Imperativa e será assinalada às autoridades académicas competentes**.

- Só são cotadas as respostas legíveis.

- A elegância e eficiência do código serão elementos de avaliação. Por exemplo, recorra à definição de funções sempre que justificado.

- Um exercício pode exigir uma determinada solução (por exemplo, *"não usar o tipo float"*, *"não usar ciclos"*, etc.).

  Qualquer solução que não respeite estas exigências será avaliada no máximo **para metade da sua cotação**.

Exercício 1 (Problema A - 2.5 pontos)

Considere a seguinte equação, parametrizada por um inteiro natural n:

$$\sum_{i=0}^{n} i^2 = \frac{n \times (n+1) \times (2 \times n + 1)}{6}$$

Como vimos em exercícios anteriores, a elegância matemática deste tipo de equações numéricas sofre um golpe quando implementadas em computadores: a artimética de computadores não é a aritmética como a vimos na escola.

O desafio: *Estamos assim interessados em saber qual é o* ***primeiro valor positivo*** *de n de tipo int para o qual esta equação deixa de ser verdade quando programada em C.Chamamos a este valor de n o valor da* ***discordância****.*

1. *Defina e apresente uma função* `C` *de assinatura* `int sum_square(int n)` *que calcule o valor de* $\sum_{i=0}^{n} i^2$ *usando explicitmente o somatório.*

2. *Defina e apresente uma função* `C` *de assinatura* `int direct(int n)` *que calcula o lado direito da equação para um determinado n.*

3. *Defina e apresente uma função* `C` *de assinatura* `int discorda(void)` *que, fazendo uso das funções das alíneas anteriores, encontra o primeiro (o menor) valor de* `n` *tal que o lado esquerdo da equação seja diferente do segundo lado da equação. É esperado que esta função devolva o valor de* `n`*.*

Como sabe, uma forma de mitigar este fenómeno é usar um tipo numérico maior, como o `long long int` *para os valores do lado direito ou do lado esquerdo da equação.*

Se tal fizermos, o valor da discordância aumenta muito. Por outro lado surge uma dificuldade diferente: o processo de cálculo leva muito, mas muito mais, tempo!

4. *Diga porque este fenómeno do grande tempo de cálculo acontece. Qual é o lado da equação culpada pelo desperdício de tempo? Porque o tempo do calculo vai piorando a medida que procuramos pelo valor da discordância para valores maiores de n?*

5. *Existe uma forma muito eficiente de calcular o valor de* $\sum_{i=0}^{n+1} i^2$ *sabendo o valor previamente calculado de* $\sum_{i=0}^{n} i^2$*. Proponha uma função* `sum_square_fast` *que use este truque para calcular explicitamente o somatório. Esta função deverá devolver um resultado de tipo* `long long int`*.*

6. *Proponha uma nova implementação da função* `discorda`*, a que chamado de* `discorda_fast`*, que tire proveito da função anterior para calcular o valor da discordância (de tipo* `long long int`*).*

□

Exercício 2 (Problema B - 2 pontos)

Pretendemos estudar a eficiência energética de uma determinada casa ao longo de uma semana. Para tal instalamos $\boxed{s}$ *sensores de temperatura pela casa toda em locais estratégicos e recolhemos a temperatura medida por estes sensores* $\boxed{m}$ *vezes durante o periodo de estudo.*

No absoluto, sabemos que

$$0 < m \leq 1000$$
$$0 < s \leq 500$$

1. *Como declara em* `C` *uma matriz mat que pode albergar os valores todos recolhidos e organizados por sensores, e medições (nesta ordem)?*

2. *Escreva uma função* `C` *de nome* `read_data` *que aceita em parâmetro uma matriz declarada e dimensionada para este problema de medição, mas também o valor de s e de m. Esta função inicializa os valores da matriz, sensor por sensor, medição por medição, com valores obtidos da entrada standard* `stdin`*.*

3. *Escreva uma função* `C` *de nome* `maior` *que devolve a maior temperatura de todas as temperaturas recolhidas pela rede de sensores nos dias todos da medição.*

4. *Escreva uma função* `media_por_sensor` *que calcula a média das temperaturas por cada sensor. As medições são passadas em parâmetro à função assim como as dimensões m e s. Espera-se que estes valores médios sejam arquivados num vector de dimensão e de tipo apropriado. Este vetor é também passado em parâmetro à função* `media_por_sensor` *.*

□

Exercício 3 (Problema C - 2.5 pontos)

Considere o seguinte problema de xadrez que involve cavalos. Relembra-se que um cavalo na posição assinalada por ***X*** *na imagem do tabuleiro pode deslocar-se para as posições (ou atacar qualquer peça que ali se encontra) assinaladas por* ***C****.*

$i-2$			C		C		
$i-1$		C				C	
i				**X**			
$i+1$		C				C	
$i+2$			C		C		

Se considerarmos que a posição $(0,0)$ *está no canto superior esquerto e que a posição do cavalo é* (x,y)*, então a posição assinalada por* ***C*** *sublinhado mais acima a direita é* $(x-2, y+1)$*.*

O desafio: *estando n cavalos colocados num tabuleiro de xadrez, o desafio deste exercício é saber detetar quando um destes cavalos pode atacar outro.*

Primeiro, assuma a seguinte definição: `#define MAX 10 //n. max de cavalos`

Vamos arquivar as posições de todos os cavalos do tabuleiro num vector declarado da seguinte forma: `int cav[2][MAX];`

	0 ...	i	... MAX-1
0	...	x	...
1	...	y	...

A posição (x,y) *do* `i`*-ésimo cavalo (com* $0 \leq$ `i` $< MAX$*) está arquivada na posição* `i` *de* `cav` *como indicado na figura.*

Para este desafio, basta-nos considerar esta matriz*, não precisamos de considerar o tabuleiro de xadrez como um todo.*

1. *Defina uma função* `C` *de nome e assinatura*

   `void ler_cavalos(int cav[2][MAX], int n);`

   Esta função contempla um vector `cav` *dimensionado para receber as posições dos* `n` *cavalos colocados no tabuleiro. O parâmetro* `n` *é garantidamente positivo não nulo e menor ou igual a* `MAX`*.*

   Espera-se que está função leia os valores para `cav` *da entrada standard (*`stdin`*).*

2. *Defina em* `C` *uma função de nome e assinatura seguinte*

   `int ataca(int c1x, int c1y, int c2x, int c2y);`

   que devolve 1 *se o cavalo* C_1 *posicionado em* `(c1x,c1y)` *consegue atacar o cavalo* C_2 *que está na posição* `(c2x,c2y)`*. Devolve* 0*, no caso contrário.*

3. *Defina em* `C` *uma função de nome e assinatura seguinte*

   `int guerra(int cav[2][MAX], int n);`

   e que verifica, ***para cada cavalo*** *de* `cav`*, se este consegue atacar os cavalos a sua direita na matriz* `cav`*. No caso de haver um ataque possível, a função devolve* `1`*, devolve* `0` *no outro caso.*

□