1º trabalho de Física Computacional (2022-23, P4)

Entrega do trabalho

- Até às 13H de dia 27 de Maio de 2023 (sábado) através do svn, **única e exclusivamente**
  - Não serão aceites trabalhos entregues por outras vias que não o svn
- Não se esqueçam de fazer `commit` de todos ficheiros com excepção dos ficheiros `*.o` `*.exe`
  - Nota: A operação `svn status` permite identificar os ficheiros ainda não `commited` ou que não estejam ainda sob controlo de svn.
- Caso existam entregas posteriores às 13H05 (atrasos superiores a 5 minutos de tolerância), a regra de desconto em valores [0,20] que será aplicada à nota do trabalho será a seguinte:

  $D = T * 0.2$
  D: desconto em valores [0,20]
  T: tempo de atraso de entrega em minutos, a partir das 13H05 (fim da tolerância)

Organização das pastas de trabalho

Na área de trabalho de cada grupo, foi criada a pasta **trab01** que contém as seguintes pastas e ficheiros (não se esqueçam de começar por fazer `svn update`):

```
trab01/src ..................... [user classes or functions]
      /main ..................... [main programs]
      /bin ...................... [object files, executables]
      Folha_Respostas.pdf ....... [folha de respostas]
      Trab01.pdf ................ [enunciado do trabalho]
      imagem.pgm ................ [ascii file with image]
```

Muito importante:

- Os programas principais que realizarem devem ser colocados na pasta `main/`
- As classes (ficheiros "header" `.h` e "source" `.C`) desenvolvidos pelo grupo e necessários à resolução deste trabalho, devem estar na pasta `src/`

Correcção e Cotação

A resolução do trabalho implicará a entrega:

- dos códigos C++ realizados (classes, funções e programa principal)
- da folha de respostas (ficheiro `Folha_Respostas.pdf`) preenchida onde constará a identificação dos elementos do grupo que participaram na realização do trabalho
  - o software PDFexpert (macOS) e Okular (linux) permitem a escrita no ficheiro pdf

```
a. como instalar o Okular no linux/ubuntu?
   sudo apt update && sudo apt upgrade -y
   sudo apt install okular -y
b. como instalar o PDFexpert no macOS?
   descarregar do site a versão gratuita: https://pdfexpert.com
```

A folha de respostas pode ser preenchida à mão ou no computador e deve ser submetida no svn. Caso seja preenchida à mão proceda à sua digitalização e junte ao svn, na pasta `trab01`.

Passos que serão seguidos na correcção do exercício:

- estrutura, clareza e comentários do código C++ (10%)
- execução do código, resultados e respostas (90%)

Enunciado

Filtragem de imagens

O enunciado do 1º trabalho da disciplina consiste em duas partes,

- uma **parte A** que corresponde ao enunciado do *Homework 1*, e que se reproduz nas páginas seguintes deste enunciado. Notem que a imagem a analisar neste trabalho é diferente da que foi usada na preparação do *homework*.

  > Os alunos devem:
  >
  > - implementar um programa C++ `rImagem` que responda às questões do *homework 1*
  > - responder às questões formuladas na Folha de Respostas

- uma **parte B** complementar ao enunciado do *Homework 1*, que corresponde às questões que se enunciam de seguida

  > Os alunos devem:
  >
  > - implementar no programa C++ `rImagem` a solução às questões da parte B
  > - responder às questões da parte B formuladas na Folha de Respostas

A folha de respostas deve ser preenchida e entregue com o trabalho.

Parte B (5 val.)

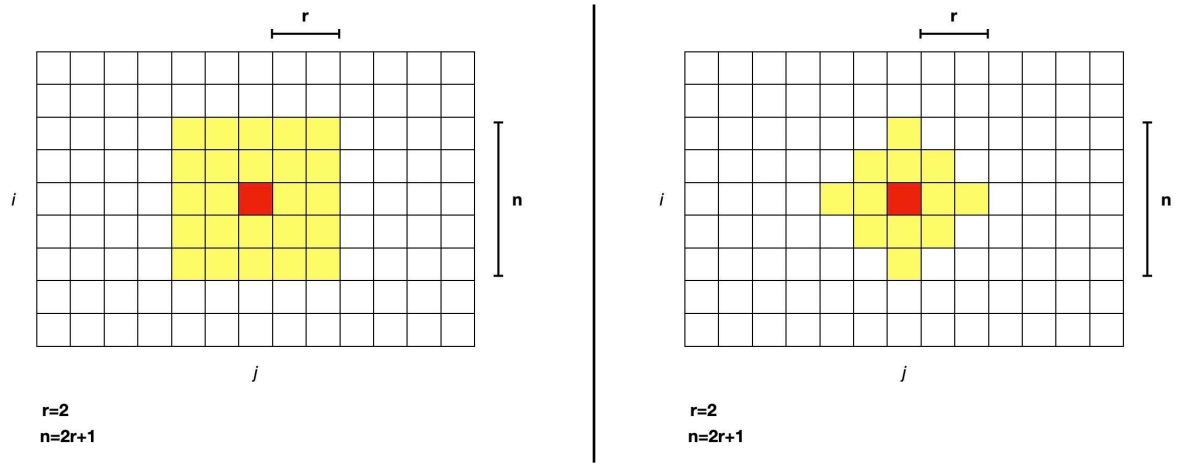
Filtragem da imagem com filtro de mediana

O processo de filtragem da imagem que aplicámos ao longo do *homework* correspondem a transformações lineares da imagem. Para imagens com ruídos de impulso como é o caso do sal e pimenta, não são os filtros mais apropriados. Os filtros mais eficazes são os filtros de mediana. Neste processo de filtragem, realizamos as seguintes operações:

- define-se um conjunto de pixeis na vizinhança do pixel a substituir (p_i), incluindo o próprio
- ordena-se o conjunto de pixeis de acordo com a sua cor e escolhemos a mediana, ou seja, a cor que fica no meio da série ordenada
- substitui-se a cor do pixel p_i pela cor do pixel da mediana

Existem várias formas possíveis de seleção dos pixeis de vizinhança, tal como se apresenta na figura ao lado. As formas geométricas definidas englobam o pixel a substituir no seu centro e outros pixeis

que cumpram um critério de distância dependendo da forma. Na resolução deste problema propõe-se a utilização das formas quadradas (figura da esquerda) e diamante (figura da direita) para a definição dos filtros de mediana.



a. Defina um filtro de mediana quadrado, com um número de pixeis de cada lado $r = 2$, e aplique-o à imagem.
   Salve a imagem filtrada num ficheiro `peixe_filtro_mediana_quadrado.ascii.pgm`.
   Determine a média e a variância da cor da imagem.

b. Defina um filtro de mediana diamante, com um número de pixeis de largura $r = 2$, e aplique-o à imagem.
   Salve a imagem filtrada num ficheiro `peixe_filtro_mediana_diamante.ascii.pgm`.
   Determine a média e a variância da cor da imagem.

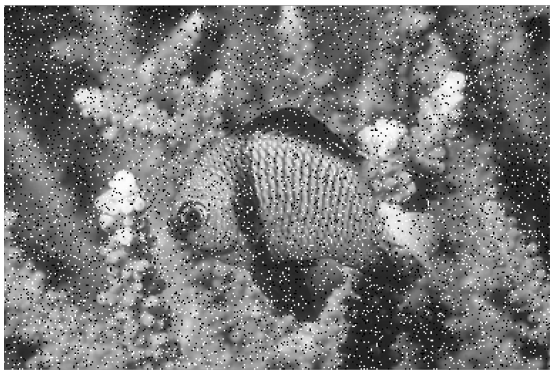
c. Diga, justificando com argumentos e computacionalmente, se teria valido a pena a aplicação de um filtro de maior largura nas alíneas a) e b). Se pensar que sim, qual o r que sugere que deveria ter sido aplicado nas alíneas anteriores?

Parte A (15 val.)

Filtragem de imagens

Actualmente, 45% dos portugueses têm uma conta Instagram ativa. De facto, cada vez que carregamos uma foto nas nossas redes sociais, a nossa maior preocupação é escolher um conjunto de filtros que melhore a qualidade das imagens. Portanto, o objetivo do exercício proposto é criar vários algoritmos que permitam melhorar a qualidade das nossas fotografias.

Para este efeito, consideraremos uma imagem **.pgm** em preto e branco. O protocolo de codificação `.pgm` atribui a cada pixel da imagem um número natural entre 0 e N. O número 0 corresponde à cor preta e por outro lado o número N corresponde à cor branca. O ficheiro `peixe_noise10.ascii.pgm` contém um exemplo de imagem codificada através do protocolo PGM.



A primeira linha do ficheiro da imagem contém a sigla *P2*, que identifica o protocolo de codificação do ficheiro. Para além disso, a segunda linha define a dimensão da imagem: no exemplo proposto, os números *384* e *256* determinam o número de pixels da fotografia ($384 \times 256 = 98304$). De facto, uma fotografia é uma matriz cujas entradas correspondem à codificação da cor dos pixéis. Portanto, os dois números armazenados na segunda linha do ficheiro PGM representam o tamanho da matriz associada à imagem.

Por outro lado, a terceira linha do ficheiro tem o papel de identificar a gradação máxima da cor N que corresponde à cor branca. Na imagem anexada, o valor correspondente ao branco será 255.

Nas linhas seguintes, os valores armazenados são as codificações que indentificam univocamente a cor de cada pixel da imagem, segundo a convenção mencionada anteriormente (0 = cor preta, N = cor branca).

Sugestão: descarreguem o ficheiro da imagem `.pgm` para o vosso computador, podendo verificar o seu conteúdo fazendo:

```
cat <ficheiro_com_a_imagem>.pgm   [enter]
```

Verifiquem que a informação guardada no ficheiro está de acordo com a explicação anteriormente fornecida.

Para a resolução deste exercício, criem um programa em C++ `rImagem.C` (podem usar a extensão .C ou outra que seja aceite como por exemplo .cpp, .cxx, .cc).

Tarefas e questões:

1. **Leitura da imagem `.pgm`**

   Introduzam no vosso programa o código C++ necessário à leitura do ficheiro da imagem e do armazenamento do seu conteúdo num objecto do tipo vector de vectores, `vector<vector<int>>` , que corresponde a uma matriz. Mostra-se de seguida um exemplo da estrutura do código C++ a implementar para esta tarefa.

```
// 1. read image configuration (3 first lines)
//... abrir o ficheiro da imagem
ifstream FI("filename");
string line;
//... ler a primeira linha
std::getline(FI, line));
std::getline(FI, line);
//... ler a 2a linha e recuperar o numero de linhas e colunas
std::istringstream iss(line);
int nrows, ncols;
iss >> ncols >> nrows;
//... ler a 3a linha e recuperar o codigo maximo de cor
std::getline(FI, line);
iss.str(line);
int WhiteValue;
iss >> WhiteValue;

// 2. creation of matrix M [nrows, ncols]
vector<vector<int>> M(nrows, vector<int>(ncols));

// 3. read image to matrix
(...)
```

   De forma a simplificar o código C++, podem autonomizar a leitura do ficheiro numa função que seja chamada a partir do programa principal. O protótipo da função (declaração da função que contém o seu nome assim como os argumentos de entrada e saída) poderia ser como se indica de seguida, em duas formas diferentes de implementação:

```
vector<vector<int>> ReadImage(string filename); // passing output by
↪  copy
void ReadImage(string filename, vector<vector<int>>& M); // passing
↪  output by reference
```

2. **Criação de um histograma de frequências absolutas das cores da imagem**

Para tal comecem por criar um `vector` de contagens com a dimensão do número de cores $N+1$, da cor preta à cor branca passando pelos tons de cinzento.

```
vector<int> ColourFreq(N+1);
```

E de seguida, armazenem no `vector` a frequência absoluta das cores existentes na imagem. Algoritmicamente, varram todos os pixéis da imagem e incrementem de 1, o contador existente na posição do `vector` correspondente a cada código de cor. De forma simbólica, calculem o número k_i de pixéis caracterizados pela cor de código i, $\forall i \in 0, ..., N$

Imprimam no ecrã o `vector` de contagens obtido. Quais as cores mais frequentes?

Calcule ainda o vector das frequências relativas em percentagem e imprima também no ecrã.

3. **Variância e média das cores da imagem**

   Para a análise da imagem podem recorrer aos estimadores estatísticos, média $\mu \equiv < x >$ e variância $Var(x) = \sigma^2(x)$, onde x é o código de cor. O primeiro informa-nos da cor média existente na imagem: quanto maior for esta cor mais clara será a imagem. O segundo quantifica a variação de cor na imagem, ou seja, a sua não uniformidade do ponto de vista da cor. O cálculo da cor média (μ) para os $N+1$ cores, pode ser calculado fazendo o varrimento de todos os pixéis e utilizando a cor de cada pixel $x_{i,j}$,

   $$\mu_x = \frac{1}{N_{pixels}} \sum_{i=1}^{nrows} \sum_{j=1}^{ncols} x_{i,j}$$

   O cálculo da variância da cor pode ser realizado da seguinte forma,

   $$Var(x) \equiv \sigma_x^2 = \frac{1}{N_{pixels} - 1} \left[\sum_{i=1}^{nrows} \sum_{j=1}^{ncols} (x_{i,j} - \mu_x)^2 \right]$$

   Imprimam no ecrã os valores da média (μ_x) e do desvio padrão (σ_x).

4. **Criação da imagem invertida**

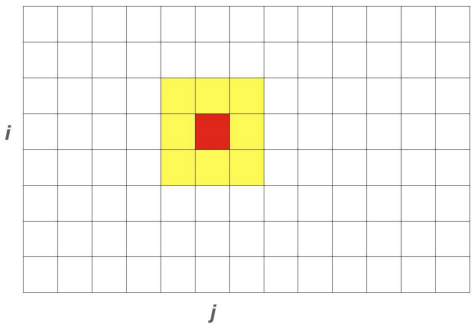
   Para este efeito, considerem que inverter uma imagem significa trocar a cor de cada pixel com a cor complementar ($0 \rightarrow N$, $1 \rightarrow N-1$, $2 \rightarrow N-2$, ...). Portanto, designando $x(i,j)$ como a cor do pixel (i,j) antes de inverter a imagem, valerá a relação $x'(i,j) = N - x(i,j)$, sendo $x'(i,j)$ a cor do pixel (i,j) após a inversão.

   Escreva num novo ficheiro, `peixe_noise10_inverted.ascii.pgm` , a imagem invertida no formato `.pgm` acima descrito.

   Somem a imagem inicial com a imagem inversa. Qual é o resultado esperado?

5. **Eliminação do ruído da imagem**

Olhando atentamente para a imagem, verificarão que esta possui perturbações de cor (ruído) relacionados com o seu envelhecimento. Nesta tarefa, vão procurar melhorar a qualidade da imagem. Para este efeito, considerem a cor de um dado pixel $x(i,j)$ da fotografia. Vamos considerar do ponto vista algorítmico, que *idealmente* a cor de um pixel está relacionado com a sua vizinhança próxima. Nesse sentido, calculem a média μ_8 dos 8 pixéis à volta do pixel $x(i,j)$ e troquem o valor de $x(i,j)$ por μ_8.



Repitam esta operação para todos os pixéis. Do ponto de vista matemático, isto corresponde a substituir a cor de cada pixel $x(i,j)$ por:

$$x'(i,j) = \frac{1}{8}\left\{\left[\sum_{n=i-1}^{i+1}\sum_{m=j-1}^{j+1} x(n,m)\right] - x(i,j)\right\}$$

Guardem a nova imagem num ficheiro `peixe_reduced_noise.ascii.pgm`.

6. **Variância da cor após eliminação do ruído**

Repitam o cálculo da variância da variável da cor da imagem e comparem o resultado com o valor obtido anteriormente na alínea 3.

7. **Filtragem da imagem: box filtering**

O algoritmo explicado na alínea 5 consegue eliminar os pontos isolados brancos e pretos da imagem. Contudo, deteriora a qualidade dos píxels que ficam à volta dos pontos isolados. De facto, o algoritmo de redução do ruído anteriormente descrito não tem em conta a cor original do pixel (i,j) que está a corrigir, podendo por isso introduzir uma deterioração da cor dos pixels na proximidade desses pontos.

Para melhorar o algoritmo, vamos introduzir a matriz de filtragem W e corrigirmos a cor dos pixels da imagem de acordo com a relação,

$$x'(i,j) = \sum_{\Delta i=-1}^{1}\sum_{\Delta j=-1}^{1} x(i+\Delta i, j+\Delta j)\, W(1+\Delta i, 1+\Delta j)$$

A definição da matriz W permite atribuir um peso diferente a cada pixel que intervém no processo de correção. Por exemplo, no caso do algoritmo da alínea 5, a matriz W possui o tamanho 3×3,

e o valor dos seus elementos são $W(1,1) = 0$ e todos os restantes elementos possuem o valor $1/8$.

Defina agora um novo algoritmo de filtragem da imagem em que atribua o valor $1/9$ (pesos constantes) a todas as entradas da matriz W.

Repitam o calculo da variância da cor e comparem com os resultados obtidos anteriormente.

Guardem a nova imagem com o nome `peixe_box_blur.ascii.pgm` .