Universidade Federal de Viçosa
Departamento de Informática
INF 112 - Programação II
2º Semestre de 2024
Professor: Julio Cesar Soares dos Reis
Estagiários de Ensino: Bruno Alencar Vieira de Rezende e Paula T. Mota Gibrim
Valor: 5 Pontos

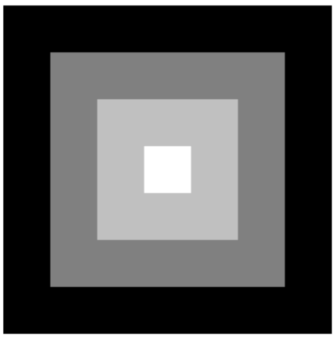
Trabalho Prático 1

Objetivos

Consiste em rever conceitos básicos de programação bem como explorar os conceitos de Ponteiros + Alocação de Memória + Tipos Abstratos de Dados (TADs).

Descrição

Você deverá implementar um tipo abstrato de dados `Imagem` para representar uma imagem quadrada em escala de cinza. O TAD `Imagem` deve armazenar a dimensão `n`, que representa tanto a altura quanto a largura da imagem. Além disso, ele deverá armazenar os valores de todos os pixels, onde cada pixel é representado por um número inteiro entre 0 e 255, indicando a intensidade de cinza (com 0 sendo preto absoluto e 255 sendo branco absoluto). Esses valores serão organizados em uma matriz, conforme ilustrado na Figura 1.



0	0	0	0	0	0	0
0	128	128	128	128	128	0
0	128	192	192	192	128	0
0	128	192	255	192	128	0
0	128	192	192	192	128	0
0	128	128	128	128	128	0
0	0	0	0	0	0	0

Figura 1: Exemplo de imagem quadrada em escala de cinza.

A dimensão da imagem será determinada em tempo de execução. Para isso, a alocação de memória deve ser feita dinamicamente. O TAD `Imagem` deverá incluir um construtor, responsável pela leitura do valor de `n` e pela alocação dinâmica da matriz de pixels, bem como um destrutor para liberar corretamente a memória. Além disso, o TAD deve oferecer funções (ou procedimentos) para:

1. `leImagem`:
   - **Entrada (1 atributo)**: referência para uma `Imagem`
   - **Tipo de retorno**: `void`
   - **Objetivo da função**: A função tem como objetivo realizar a leitura dos dados de uma imagem quadrada em escala de cinza. Ela deve solicitar ao usuário o valor de `n` (dimensão da imagem) e, em seguida, solicitar os valores dos pixels da imagem, que variam entre 0 e 255. Caso o usuário insira um valor superior a 255, ele deverá ser transformado em 255; se for inferior a 0, deverá ser transformado em 0. A função também deve garantir a alocação dinâmica da memória necessária para armazenar esses dados.
2. `soma`:
   - **Entrada (3 atributos)**: duas referências para `Imagem` (as imagens de entrada) e uma referência para `Imagem` (a imagem resultante)
   - **Tipo de retorno**: `void`

- **Objetivo da função**: A função deve somar pixel a pixel duas imagens quadradas em escala de cinza. O valor de cada pixel da imagem resultante deve ser a soma dos pixels correspondentes das duas imagens de entrada. Caso a soma resulte em um valor superior a 255, o pixel deve ser ajustado para 255. A função deve garantir que a memória da imagem resultante já esteja alocada dinamicamente antes da operação. Caso as dimensões das imagens sejam diferentes, a função deve apenas imprimir uma mensagem de erro, sem realizar nenhuma alteração.

3. `subtrai`:

   - **Entrada (3 atributos)**: duas referências para `Imagem` (as imagens de entrada) e uma referência para `Imagem` (a imagem resultante)
   - **Tipo de retorno**: `void`
   - **Objetivo da função**: A função deve subtrair pixel a pixel duas imagens quadradas em escala de cinza. O valor de cada pixel da imagem resultante deve ser a diferença entre os pixels correspondentes das duas imagens de entrada. Caso a subtração resulte em um valor negativo, o pixel deve ser ajustado para 0. A função deve garantir que a memória da imagem resultante já esteja alocada dinamicamente antes da operação. Caso as dimensões das imagens sejam diferentes, a função deve apenas imprimir uma mensagem de erro, sem realizar nenhuma alteração.

4. `criaImagemPreta`:

   - **Entrada (2 atributos)**: dimensão `n` e uma referência para `Imagem` (a imagem resultante)
   - **Tipo de retorno**: `void`
   - **Objetivo da função**: A função deve criar uma imagem quadrada de dimensão `n`, onde todos os pixels terão o valor 0, representando uma imagem completamente preta. A função deve garantir que a memória para a imagem resultante seja alocada dinamicamente e configurada corretamente.

5. `inverteImagem`:

   - **Entrada (1 atributo)**: referência para uma `Imagem`
   - **Tipo de retorno**: `void`
   - **Objetivo da função**: A função deve inverter os valores de todos os pixels de uma imagem quadrada em escala de cinza. Para cada pixel, o valor invertido deve ser `255 - valor_original`. A função não cria uma nova imagem, mas altera os valores da imagem existente.

6. `compara`:

   - **Entrada (2 atributos)**: duas referências para `Imagem`
   - **Tipo de retorno**: `bool`
   - **Objetivo da função**: A função deve comparar pixel a pixel duas imagens quadradas. Se todas as dimensões e os valores dos pixels forem idênticos, a função deve retornar `true`; caso contrário, deve retornar `false`. Caso as dimensões das imagens sejam diferentes, as imagens não são iguais.

7. `imprimeImagem`:

   - **Entrada (1 atributo)**: referência para uma `Imagem`
   - **Tipo de retorno**: `void`
   - **Objetivo da função**: A função deve imprimir os valores dos pixels de uma imagem quadrada em escala de cinza, onde cada linha corresponde a uma linha de pixels da imagem. O formato de exibição pode ser uma matriz de inteiros, conforme exemplificado à direita na Figura 1.

8. `adicionaBorda`:

   - **Entrada (1 atributo)**: referência para uma `Imagem`
   - **Tipo de retorno**: `void`
   - **Objetivo da função**: A função adiciona uma borda de 2 pixels ao redor da imagem. A camada externa da borda deve ser preta (pixels com valor 0) e a interna, branca (pixels com valor 255). Se a imagem já tiver uma borda branca, será necessário apenas adicionar uma camada preta de 1 pixel ao redor. A Figura 2 exemplifica a execução dessa função.

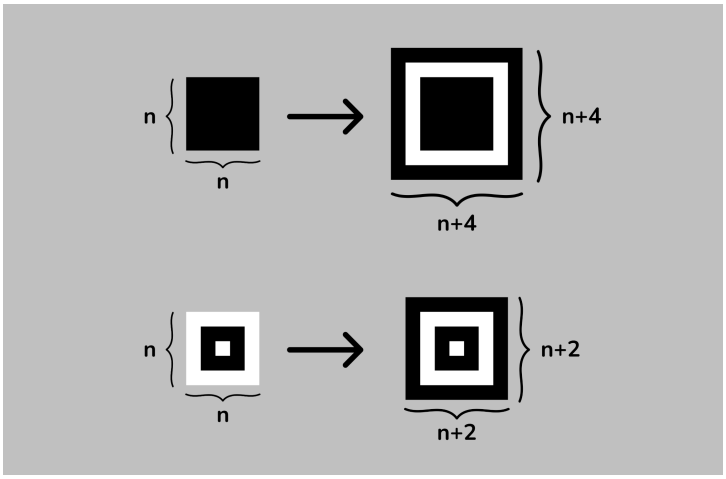


Figura 2: Exemplo da execução da função `adicionaBorda`.

O TAD deve armazenar os seguintes elementos:

1. `n`: a dimensão da imagem, que representa tanto a altura quanto a largura.
2. `pixels`: uma matriz de inteiros `n x n` implementada como `int **`, onde cada elemento representa um pixel com valores de 0 a 255.

O seu programa deve estar modularizado (.h e .cpp) e além disso, deve conter um `main.cpp`, que utilize o seu TAD. O seu programa deverá ter uma estrutura **similar**[1] a apresentada abaixo, contendo **obrigatoriamente** chamada para todas as funções implementadas.

```
/* ... Declaracao e implementacao do TAD Imagem ... */
int main( ) {
  Imagem img1, img2, imgSomada, imgSubtraida, imgInvertida, imgPreta, imgBranca;

  // Leitura de imagens
  leImagem(img1); // Le uma imagem quadrada em escala de cinza (uma matriz de inteiros nxn)
  leImagem(img2); // Le uma segunda imagem

  // Somando duas imagens
  soma(img1, img2, imgSomada); // imgSomada = img1 + img2
  std::cout << "Imagem Somada:" << std::endl;
  imprimeImagem(imgSomada); // Exibe a imagem resultante da soma

  // Subtraindo duas imagens
  subtrai(img1, img2, imgSubtraida); // imgSubtraida = img1 - img2
  std::cout << "Imagem Subtraida:" << std::endl;
  imprimeImagem(imgSubtraida); // Exibe a imagem resultante da subtracao

  // Invertendo uma imagem
  inverteImagem(img1); // Inverte os pixels da img1
  std::cout << "Imagem Invertida (img1):" << std::endl;
  imprimeImagem(img1); // Exibe a imagem invertida

  // Comparando duas imagens
  if (compara(img1, img2)) {
    std::cout << "Imagens iguais." << std::endl;
  } else {
    std::cout << "Imagens diferentes." << std::endl;
  }

  // Criando uma imagem preta
  criaImagemPreta(5, imgPreta); // Cria uma imagem preta de dimensao 5x5
  std::cout << "Imagem Preta:" << std::endl;
  imprimeImagem(imgPreta); // Exibe a imagem preta
```

[1] Isso significa que adequações podem/devem ser efetuadas. Este é apenas um exemplo para direcionamento.

```
  adicionaBorda(imgPreta); // Adiciona borda na imagem preta (novo tamanho => 9x9)
  std::cout << "Imagem Preta com Borda:" << std::endl;
  imprimeImagem(imgPreta); // Exibe a imagem preta com borda

  // Criando uma imagem branca
  criaImagemPreta(5, imgBranca); // Cria uma imagem preta de dimensao 5x5
  inverteImagem(imgBranca); // Inverte os pixels da imagem preta, tornando-os brancos
  std::cout << "Imagem Branca:" << std::endl;
  imprimeImagem(imgBranca); // Exibe a imagem branca
  adicionaBorda(imgBranca); // Adiciona borda na imagem branca (novo tamanho => 7x7)
  std::cout << "Imagem Branca com Borda:" << std::endl;
  imprimeImagem(imgBranca); // Exibe a imagem branca com borda


  return 0;
}
```

Exemplo

Para o código acima, um possível exemplo seria:

- **Entrada**:

```
5
100 150 200 250 100
50 75 100 125 150
255 200 150 100 50
0 25 50 75 100
200 150 100 50 0
5
30 60 90 120 150
180 210 240 30 60
255 225 195 165 135
105 75 45 15 255
200 170 140 110 80
```

- **Saída** (foram adicionadas quebras de linha adicionais para maior clareza):

```
Imagem Somada:
130 210 255 255 250
230 285 340 155 210
255 255 255 255 185
105 100 95 90 255
255 255 240 160 80

Imagem Subtraida:
70 90 110 130 0
0 0 0 95 90
0 0 0 0 0
0 0 5 60 0
0 0 0 0 0

Imagem Invertida (img1):
155 105 55 5 155
205 180 155 130 105
0 55 105 155 205
255 230 205 180 155
55 105 155 205 255

Imagens diferentes.

Imagem Preta:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

```
0 0 0 0 0
0 0 0 0 0

Imagem Preta com Borda:
0 0 0 0 0 0 0 0 0
0 255 255 255 255 255 255 255 0
0 255 0 0 0 0 0 255 0
0 255 0 0 0 0 0 255 0
0 255 0 0 0 0 0 255 0
0 255 0 0 0 0 0 255 0
0 255 0 0 0 0 0 255 0
0 255 255 255 255 255 255 255 0
0 0 0 0 0 0 0 0 0

Imagem Branca:
255 255 255 255 255
255 255 255 255 255
255 255 255 255 255
255 255 255 255 255
255 255 255 255 255

Imagem Branca com Borda:
0 0 0 0 0 0 0
0 255 255 255 255 255 0
0 255 255 255 255 255 0
0 255 255 255 255 255 0
0 255 255 255 255 255 0
0 255 255 255 255 255 0
0 0 0 0 0 0 0
```

Dicas

- Não se esqueça de liberar os espaços de memória alocados dinamicamente para evitar vazamentos de memória. Para verificar se todos os espaços foram liberados corretamente, você pode utilizar a ferramenta de depuração **Valgrind** (https://valgrind.org).
- Indente e adicione comentários ao seu código. Isso facilita a leitura e compreensão, tanto para você quanto para outras pessoas que possam revisá-lo.
- Reutilize código! Sempre que precisar realizar a mesma tarefa em diferentes partes do programa, considere criar uma função específica para essa ação. Por exemplo, pode ser útil implementar uma função que receba um valor inteiro e ajuste-o automaticamente para permanecer entre 0 e 255, garantindo que os valores dos pixels estejam sempre dentro da faixa permitida.

Desenvolvimento e Entrega

O código fonte do programa deve ser desenvolvido em C++, estar bem identado e comentado. A entrega deve ser efetuada conforme agendado no PVANet Moodle. Para isso, você deve criar um projeto contendo os arquivos `.h`, `.cpp`, e `main.cpp` criados. Envie, através do PVANet Moodle, uma pasta compactada (.rar ou .zip) contendo o projeto. A pasta compactada deve conter informações do aluno (ex.: julio_reis-tp1.zip). Para correção, serão considerados os seguintes critérios:

1. Documentação (**1pt**).
   (a) Detalhamento do código.
   (b) Comentários, indentação.
2. Funcionamento correto (**2 pts**).
   (a) Compila e executa, não apresenta *crash*, etc.
3. Aplicação correta dos conceitos (**2 pts**).
   (a) Uso de ponteiros, gerenciamento de memória, etc.

Comentários Gerais

- Comece a fazer este trabalho logo: o prazo para terminá-lo está tão longe quanto jamais poderá estar! :)
- O trabalho é individual (grupo de UM aluno);
- Trabalhos copiados serão penalizados (NOTA Zero).