Projecto de programação I

Programação (L.EIC009)

Eduardo R. B. Marques, DCC/FCUP

Sumário

Neste projecto terá de definir classes C++ para representar e processar imagens com cores RGB (Red-Green-Blue). As imagens podem ser carregadas ou gravadas no formato PNG usando código já fornecido. As classes são:

- `rgb::color` para representar cores;
- `rgb::image` para representar imagens;
- e `rgb::script` para processar "scripts" de texto com comandos de manipulação de imagens.

É fornecido um projecto CMake para desenvolvimento com a estrutura detalhada abaixo neste documento, e uma versão com programas solução (binários) no Replit.

Realização e entrega

O trabalho pode ser realizado individualmente ou em grupos de 2 alunos e ser entregue até **23 de Dezembro de 2021**.

1. Preencha o arquivo `README.md` identificando o grupo e fazendo um sumário das tarefas que conseguiu completar ou não.

2. Gere um arquivo ZIP para entrega executando no directório de “build” o seguinte comando:

```
make package_source
```

Confirme que é gerado um ficheiro `delivery.zip` contendo o ficheiro `README.md` e o código fonte do directório `rgb`. Se tiver problemas neste passo gere manualmente um arquivo ZIP com o mesmo conteúdo.

3. A forma de submissão do arquivo ZIP será anunciada posteriormente.

Projecto CMake

Ficheiro/directório	Descrição
`CMakeLists.txt`	**Configuração do projecto CMake.**
`rgb`	**Código fonte.**
`rgb/color.hpp`	“Header” para classe `rgb::color` (**tarefa: documentar com anotações Doxygen**).
`rgb/color.cpp`	Implementação da classe `rgb::color` (**tarefa: completar o código**).
`rgb/image.hpp`	“Header” para classe `rgb::image` (**tarefa: documentar com anotações Doxygen**).
`rgb/image.cpp`	Implementação da classe `rgb::image` (**tarefa: completar o código**).
`rgb/script.hpp`	“Header” para classe

	`rgb::script` (**tarefa: documentar com anotações Doxygen**).
`rgb/script.cpp`	Implementação da classe `rgb::script` (**tarefa: completar o código**).
`png`	**Suporte para carregamento e gravação de ficheiros PNG (não modificar).**
`png/png.hpp`	“Header” file com declaração de funções.
`png/png.cpp`	Implementação de funções.
`test`	**Testes unitários (usando GoogleTest).**
`test/color_test.cpp`	Testes para a classe `rgb::color`.
`test/image_test.cpp`	Testes elementares para a classe `rgb::image`.
`test/script_test.cpp`	Testes para a classe `image` e `script`, usando scripts e imagens fornecidas no directório `data`.
`data`	**Pasta com exemplos para processamento de scripts.**
`data/scripts`	Scripts exemplo usados por `test/script_test.cpp` e que podem também ser invocados com programa `programs/rgb_script.cpp`.
`data/input`	Imagens exemplo lidas pelos scripts.

`data/output`	Directório para imagens produzidas pela execução de scripts.
`data/expected`	Directório para imagens de validação usadas por `test/script_test.cpp`. As imagens em `output` deverão ser equivalentes às imagens correspondentes neste directório.
`programs`	**Programas utilitários (poderão ser convenientes para desenvolvimento/debugging). Não funcionarão correctamente se pelo menos testes de `image_test` não passarem.**
`programs/run_script.cpp`	Executa um script.
`programs/image_diff.cpp`	Compara duas imagens.
`programs/image_dump.cpp`	Lista valores de pixeis para uma imagem.
`external`	**Bibliotecas auxiliares.**
`external/gtest`	Código fonte da biblioteca GoogleTest, usada para programação de testes.
`external/stb`	Código fonte da biblioteca stb, usada para para ler e escrever imagens no formato PNG.

Critérios de avaliação

1. O seu código deverá estar correcto atendendo aos requisitos detalhados nas próximas páginas para as classes `color`, `image` e `script`.

   - Use os programas de teste fornecidos para validar a sua implementação.
   - Além dos aspectos funcionais, o código não deverá apresentar deficiências no uso de memória, por exemplo em termos de “buffer overflows”, “dangling references”, ou “memory leaks”.

2. O seu código deverá estar bem estruturado e ser tão simples quanto possível.
3. O código deve ser documentado no formato Doxygen.

   - Insira os comentários de documentação nos “header files” `rgb/color.hpp`, `rgb/image.hpp`, `rgb/script.hpp` para todas as declarações (`public` ou `private`).
   - Use `make docs` no directório de “build” para gerar a documentação. A documentação gerada será depositada no sub-directório `html`.

4. O código de todos os grupos será analisado por uma ferramenta de detecção automática de cópia. Trabalhos com similaridades para além da dúvida razoável serão anulados.

Como começar …

Sugere-se o seguinte fluxo de trabalho para a implementação de código:

1. Comece por completar a classe `rgb::color` até que os testes de `color_test` passem todos.
2. Complete de seguida código necessário na classe `rgb::image` até que os testes de `image_test` passem todos.
3. Complete progressivamente o resto do código em `rgb::image` e

`rgb::script` usando os testes em `script_test`.

Classe `rgb::color`

Um objecto `rgb::color` tem associados três valores inteiros (r, g, b) entre 0 e 255 para as gamas vermelha (“red”), verde (“green”), e azul (“Blue”).

Em `color.cpp` está definida a implementação da classe, que deverá ser completada. Use o programa de teste `color_test` para validar a sua implementação.

Campo ou função	Descrição
`WHITE`, `BLACK`, `RED`, `GREEN`, `BLUE`	Algumas constantes para cores comuns (**já definidos/implementados em `color.hpp/cpp`**).
`rgb_value r, g, b`	Campos membro para representar compomentes RGB (**já definidos em `color.hpp`**)
`color()`	Construtor por omissão. Deve inicializar todas as componentes RGB com valor `0`.
`color(rgb_value red,rgb_value green,rgb_value blue)`	Construtor que usa valores fornecidos para inicializar components RGB.
`color(const color& c)`	Construtor por cópia.

`rgb_value red() const`
`rgb_value& red()`
`rgb_value green() const`
	Obtêm valor ou referência para cada uma das componentes RGB (**código já implementado em `color.cpp`**).

`rgb_value& green()`
`rgb_value blue()`
`const`
`rgb_value& blue()`	

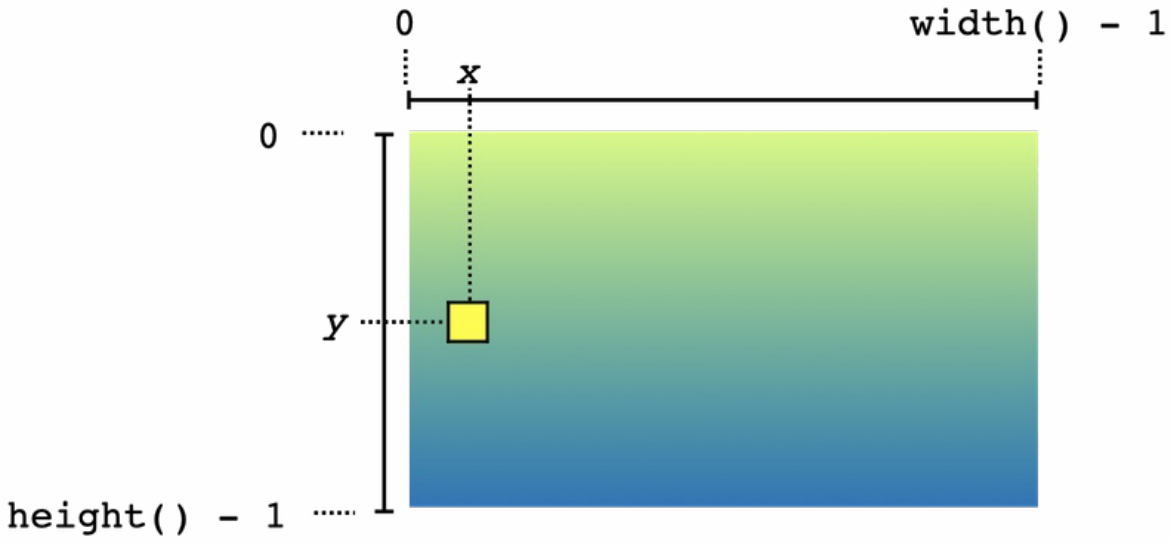
`color& operator=` `(const color& c)`	Operador de atribuição.

`bool operator==`
`(const color &c)`
`const`
`bool operator!=`
`(const color &c)`
`const`
	Operadores de comparação.

`void invert()`	Inverte cor: $(r, g, b) \mapsto (255 - r, 255 - g, 255 - b)$
`void` `to_gray_scale()`	Converte cor para uma escala de cinzento: $(r, g, b) \mapsto (v, v, v)$ onde $v = (r + g + b) / 3$.
`void mix(const` `color& c,int f)`	Mistura com cor `c` com um factor `f` entre `0` e `100`: $(r, g, b) \mapsto (m(r, c.r), m(g, c.g), m(b, c.b))$ onde $m(a, b) = ((100 - f)\, a + f\, b) / 100$.

Classe `rgb::image`

Um objecto `rgb::image` é definido por uma largura (`width()`), uma altura `height()`, e uma matriz de cores RGB `width()` × `height()` em que cada posição `(x,y)` é chamada de pixel. O canto superior esquerdo da imagem corresponde ao pixel `(0,0)` e o campo inferior direito ao pixel `(width()-1, height()-1)` como ilustrado na seguinte imagem:



Os campos e funções de `rgb::image` são detalhados na seguinte tabela.

Campo ou função	Descrição
`int iwidth`	Campo para largura da imagem (**já definido em `image.hpp`**).
`int iheight`	Campo para altura da imagem (**já definido em `image.hpp`**).
`color** pixels`	Campo para pixeis da imagem, uma matriz de cores implementada como array de apontadores (**já definido em `image.hpp`**).
`image(int w, int h, const color& fill)`	Construtor de imagem com largura `w`, altura `h` e cor inicial `fill` para todos os pixeis.
`int width() const`	Obtém largura da imagem.
`~image()`	Destructor. Deve libertar o espaço alocado para a matriz de pixeis.

`int height() const`	Obtém altura da imagem.
`color& at(int x, int y)` `const color& at(int x, int y) const`	Obtém referência mutável ou constante para cor do pixel na posição (x, y) para $0 \leq x <$ `width()` e $0 \leq y <$ `height()` **(código já implementado em `image.cpp`)**.
`void invert()`	Inverte todos os pixeis da imagem, aplicando `color.invert()`.
`void to_gray_scale()`	Converte todos os píxeis para uma escala de cinzento, aplicando `color.to_gray_scale()`.
`void replace(const color& a, const color& b)`	Altera a cor de todos os pixeis com cor `a` para a cor `b`.
`void fill(int x, int y, int w, int h, const color& c)`	Define a cor `c` para todos os pixeis com coordenadas (x', y') tais que $x \leq x' < x + w$ e $y \leq y' < y + h$.
`void mix(const image& img, int f)`	Altera imagem, misturando cada pixel com o pixel correspondente em `img` com factor `f` aplicando `color.mix()`.
`void add(const image& img, const color& neutral, int x, int y)`	Adiciona o conteúdo de `img`, substituindo os actuais pixeis a partir da posição `(x,y)` excepto para pixeis em `img` que tiverem cor `neutral`

	(cor “neutra” na “adição”).
`void crop(int x, int y, int w, int h)`	Reduz a imagem (alterando sua dimensão) ao rectângulo com o topo superior esquerdo `(x,y)` e dimensão `w` × `h`.
`void rotate_left()`	Roda imagem para a esquerda.
`void rotate_right()`	Roda imagem para a direita.

Notas importantes:

- Para definir ou libertar a matriz de pixeis, use `new` e `delete`. Pode seguir a estratégia do exercício 5 da Ficha 6.
- Em `crop`, `rotate_left` e `rotate_right` deverá re-definir os campos membro apropriadamente incluindo a matriz de pixeis.

Classe `rgb::script`

A classe `rgb::script` serve para processar scripts de texto com comandos de manipulações de imagem.

- O esqueleto já dado para classe indica que existe uma imagem em contexto para os comandos (o campo membro `img`) sobre os quais incidirão os comandos, inicialmente não definida (i.e., `NULL`).
- Os comandos `load` ou `blank` inicializam de fresco a imagem, e o comando `save` grava a imagem num ficheiro PNG.
- Os outros comandos alteram a imagem invocando métodos da classe `rgb::image`.

Comando	Descrição
`blank w h r g b`	Inicializa imagem com dimensões `w` × `h` e

	cor inicial `(r,g,b)` para todos os pixeis **(já implementado)**.
`open filename`	Inicializa imagem de ficheiro `filename` **(já implementado)**.
`save filename`	Salva imagem para ficheiro `filename` **(já implementado)**.

`invert`
`to_gray_scale`
`replace r1 g1 b1 r2 g2 b2`
`fill x y w h r g b`
`crop x y w h`
`rotate_left`
`rotate_right`
	Aplica transformações correspondentes à imagem usando funções membro correspondentes de `rgb::image`.

`mix filename f` `add filename r g b x y`	Aplica transformações à imagem da forma descrita para `rgb::image`, lendo a imagem para a mistura ou adição a partir de `filename`. Use a função `png::load` para carregar uma imagem como exemplificado para o comando `open`.

Exemplo de script (ver `data/scripts/extra4.txt`):

```
blank 750 380
      0 0 0
fill 0 0
     250 380
     255 0 0
add input/lion.png
     255 255 255
     0 0
fill 250 0 250 380
     0 255 0
```

```
add input/lion.png
      255 255 255
      250 0
fill 500 0
      250 380
      0 0 255
add input/lion.png
      255 255 255
      500 0
save output/extra4.png
```

Resultado esperado (ver `data/expected/extra4.png`):