Introdução à Programação - 1.o semestre de 2021

Terceiro Exercício Programa

Desenho de primitivas gráficas

Introdução

Neste EP3 vocês devem implementar o desenho de algumas primitivas gráficas e funções de desenho, além de completar o esqueleto do programa fornecido que gera imagens a partir de uma sequência de comandos fornecidos ao programa. O objetivo deste EP é exercitar a manipulação de matrizes (vetores bidimensionais), e imagens digitais tem tudo a ver com isso!

Imagens digitais nada mais são do que matrizes, onde cada elemento representa um pixel da imagem, e o valor de cada pixel representa a intensidade luminosa e/ou cor do pixel. O intervalo dos valores que cada pixel pode assumir depende da representação utilizada e também do tipo da imagem (colorida ou escala de cinza).

Uma representação comum para imagens em escala de cinza considera o valor inteiro zero para representar um pixel completamente preto, o valor 255 para representar um pixel totalmente branco, e valores intermediários para representar tonalidades de cinza (a informação de cada pixel pode ser, portanto, representada por 1 byte - ou 8 bits).

Já para imagens coloridas, é comum representar cada pixel por 3 valores inteiros no intervalo [0, 255]. Cada um destes três valores representa a intensidade luminosa para as componentes de cor R (*red*, vermelho), G (*green*, verde) e B (*blue*, azul) que, combinadas, podem representar em torno de 16 milhões de cores/intensidades diferentes.

Para este EP, iremos considerar imagens em escala de cinza. Logo, apenas um valor numérico no intervalo [0, 255] está associado a cada pixel da imagem, ou equivalentemente, a cada elemento da matriz. Apesar de ser possível usar uma matriz de *char* para representar uma imagem, neste EP utilizaremos uma matriz de *int* para tal finalidade[1].

Além disso, consideramos a seguinte estrutura (*struct*) para representar uma imagem, algo conveniente uma vez que agrega em um único tipo de dado a matriz que efetivamente armazena o conteúdo (pixels) da imagem, assim como suas dimensões (lembrem-se que vetores/matrizes nada mais são do que ponteiros, ou seja, guardam o endereço inicial de

[1] Uma eventual modificação para suportar imagens coloridas seria relativamente fácil de fazer neste caso, uma vez que seria possível usar um valor int para “empacotar” até 4 valores distintos no intervalo [0, 255], o suficiente para representar as componentes R, G e B do pixel, e eventualmente uma quarta componente de transparência (*alpha channel*).

blocos de memória, mas não temos acesso à informação sobre os tamanhos destes blocos):

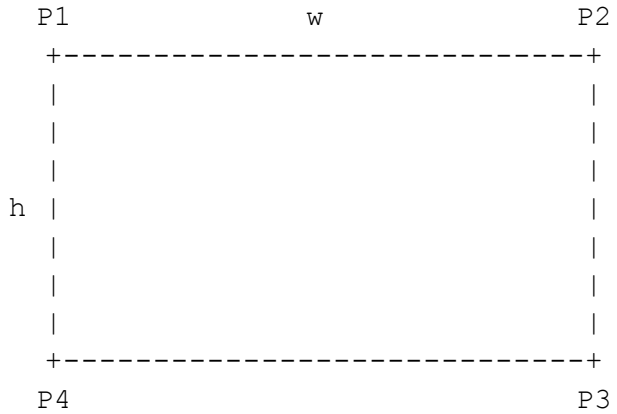
```
typedef struct {

  int altura, largura;
  int ** matriz;

} Imagem;
```

Consideramos que altura da imagem corresponde à quantidade de linhas da matriz, enquanto a largura corresponde à quantidade de colunas. Acessar um pixel da matriz usando os índices (i, j), corresponde a acessar o pixel da *linha i* (coordenada vertical) e *coluna j* (coordenada horizontal). Prestem atenção ao fato de que usualmente representamos um ponto no espaço 2D através de coordenadas (x, y), onde x representa a componente horizontal do ponto, e y sua componente vertical. Assim, o pixel da matriz associado a um ponto 2D de coordenadas (x, y) deve ser acessado pelos índices (y, x), uma vez que a coordenada vertical corresponde à linha da matriz e a coordenada horizontal corresponde à coluna.

Para entender melhor esta correspondência entre coordenadas e índice da matriz, considere o diagrama abaixo, em que os 4 cantos de uma imagem com *w* pixels de largura e *h* pixels de altura (representada, portanto, por uma matriz de *h* linhas e *w* colunas) são destacados. Seja P1 o canto superior esquerdo da imagem, P2 o canto superior direito, P3 o canto inferior direito, e P4 o canto inferior esquerdo da imagem:



As coordenadas destes pontos expressas em termos de coordenadas bidimensionais (componente horizontal, componente vertical) seriam:

- P1: (0, 0)
- P2: (w - 1, 0)
- P3: (w - 1, h - 1)
- P4: (0, h - 1)

Note que o eixo y, cresce para baixo, de modo diferente ao plano cartesiano típico ao qual estamos acostumados. Já os índices (linha, coluna) usados para acessar a informação destes pontos da matriz que representa a imagem seriam:

- P1: (0, 0)
- P2: (0, w - 1)
- P3: (h - 1, w - 1)
- P4: (h - 1, 0)

Além da estrutura definida para representar uma imagem, consideramos ainda a seguinte estrutura representar pontos no espaço 2D (note que o campo x representa a coordenada horizontal do ponto, enquanto a coordenada y a componente vertical):

```
typedef struct {

      int x, y;

} Ponto;
```

Funções a serem implementadas

Vocês devem completar o esqueleto do programa fornecido de modo a implementar as seguintes funções:

```
void reta(Imagem * imagem, Ponto2D p1, Ponto2D p2, int cor);
void retangulo_contorno(Imagem * imagem, Ponto2D p1, Ponto2D p2, int cor);
void retangulo_preenchido(Imagem * imagem, Ponto2D p1, Ponto2D p2, int cor);
void clona(Imagem * imagem, Ponto2D p1, Ponto2D p2, Ponto2D p3);
void clona_inverte_cor(Imagem * imagem, Ponto2D p1, Ponto2D p2, Ponto2D p3);
void clona_espelha_horizontal(Imagem * imagem, Ponto2D p1, Ponto2D p2, Ponto2D p3);
void clona_espelha_vertical(Imagem * imagem, Ponto2D p1, Ponto2D p2, Ponto2D p3);
```

Todas as funções recebem um ponteiro do tipo **`Imagem`** (parâmetro **`imagem`**), alguns parâmetros adicionais, e irão, cada uma à sua maneira, alterar o conteúdo da matriz que armazena o conteúdo da imagem digital.

As três primeiras funções são responsáveis pelo desenho de algumas primitivas gráficas. A função **`reta`** deve desenhar uma reta que tem origem em **`p1`** e termina em **`p2`**, com a tonalidade definida pelo parâmetro **`cor`**. Já a função **`retangulo_contorno`** deve desenhar o contorno de um retângulo definido pelos cantos opostos **`p1`** e **`p2`**, com a tonalidade de cinza definida em **`cor`**. Finalmente, a função **`retangulo_preenchido`** deve preencher toda a área retangular definida pelos cantos opostos **`p1`** e **`p2`** com a tonalidade de cinza definida em **`cor`**.

As quatro demais funções são variações de funções que clonam (copiam) o conteúdo de uma região da imagem em outra. A região a ser copiada é delimitada pelos pontos **`p1`** (canto superior esquerdo da região) e **`p2`** (canto inferior direito), e os pixels desta região devem ser copiados para uma nova região da imagem que tem como canto superior esquerdo o ponto **`p3`**. A primeira versão da função (**`clona`**) faz uma cópia idêntica dos pixels da região de origem para a região destino. Já a segunda versão (**`clona_inverte_cor`**), realiza a cópia invertendo a "cor" dos pixels (ou seja, pixels brancos se tornam pretos, pixels pretos se tornam brancos, e pixels cinza assumem o valor complementar que, somado ao valor original, resulta em 255) gerando, portanto, na região destino um "negativo" da região de origem. Por fim, as duas últimas variações (**`clona_espelha_horizontal`** e **`clona_espelha_horizontal`**) copiam os pixels da região de origem para a região destino fazendo um espelhamento na horizontal ou na vertical, respectivamente.

Programa Principal (função `main`) e função `cria_imagem`

Além de implementar as funções listadas na seção anterior deste enunciado responsáveis pelas funcionalidades de desenho, vocês também devem completar outras duas funções: a função `main`, que implementa o programa principal, e a função `cria_imagem`, responsável por inicializar uma estrutura do tipo `Imagem`, incluindo a alocação dinâmica da matriz que armazena o conteúdo (valores dos pixels) da imagem em si.

Em relação ao programa principal, este deve ler uma sequência de parâmetros/operações passados ao programa através da entrada padrão, e gerar como saída a imagem resultante após a realização das operações de desenho especificadas. Detalhando um pouco mais, o programa deve:

1) ler uma string que definirá o nome do arquivo de saída (ou seja, nome do arquivo de imagem que será criado pelo programa).
2) ler dois valores inteiros que irão determinar a largura e altura da imagem a ser gerada.
3) ler uma sequência de operações de desenho que serão executadas através das chamadas às funções correspondentes. Cada operação é expressa por uma string (que representa a operação em si) seguida de parâmetros do tipo inteiro. Este processo se repete até que a string "FIM" seja fornecida ao programa.
4) criar o arquivo da imagem resultante.
5) liberar recursos de memória que foram alocados dinamicamente.

Com exceção do item 3, os demais passos já estão implementados no esqueleto. Desta forma, o que falta à função `main` é a codificação da lógica que verifica qual a operação de desenho lida pela entrada padrão, a leitura dos parâmetros necessários para cada operação, e a chamada da função de desenho adequada.

Uma dica extremamente útil, que facilita a utilização do programa, é criar arquivos de texto contendo toda a informação que gostaríamos de passar ao programa, e redirecionar o

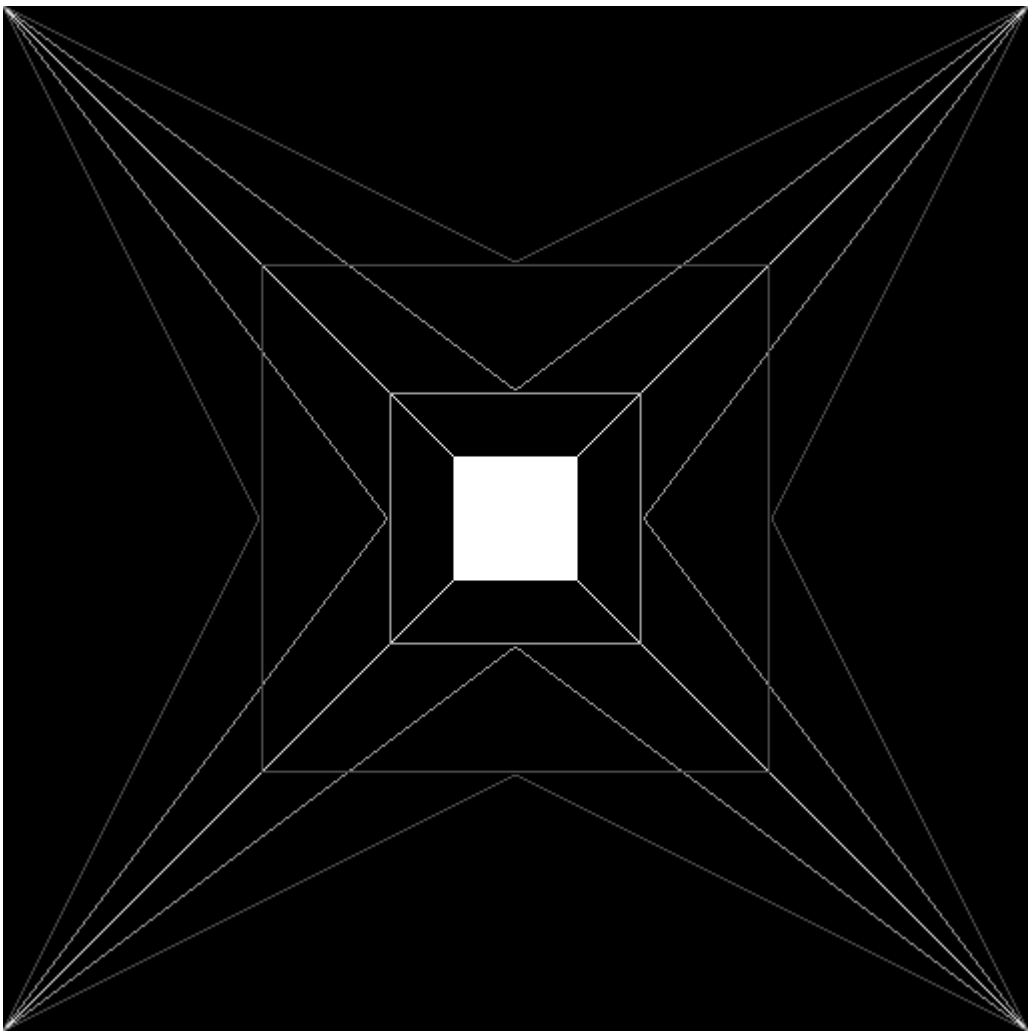
conteúdo deste arquivo para a entrada padrão do programa, executando-o da seguinte forma na linha de comando:

```
./ep3 < entrada.txt
```

Um exemplo de arquivo de entrada pode ser visto nas linhas abaixo:

```
imagem.pgm 512 512
RETA 0 0 255 255 255
RETA 0 0 127 255 80
RETA 0 0 255 127 80
RETA 0 0 191 255 160
RETA 0 0 255 191 160
CLONA_VER 0 0 255 255 0 256
CLONA_HOR 0 0 255 255 256 0
CLONA_VER 256 0 511 255 256 256
RETANGULO_CONTORNO 129 129 382 382 80
RETANGULO_CONTORNO 193 193 318 318 160
RETANGULO_PREENCHIDO 225 225 286 286 255
FIM
```

Ao executar o EP redirecionando o conteúdo deste arquivo para a entrada padrão, a seguinte imagem será produzida:



Entrega

A entrega deste EP deve ser composta apenas pelo arquivo fonte (.c) do programa desenvolvido. A entrega pode ser feita até o dia 11/08 às 23:59 pelo eDisciplinas. Não se esqueça de indicar (através de um comentário) seu nome e número USP no início do arquivo. Fique livre para adicionar outros comentários que julgar relevantes para o bom entendimento do código. Este EP pode ser feito de forma individual ou em dupla (neste caso, não se esqueça de indicar os nomes e números USP de ambos os componentes da dupla).

Boa diversão! :)