Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico de Informática (DAINF)

Fundamentos de Programação I

Profª: Juliana de Santi (jsanti@utfpr.edu.br)

Trabalho Prático - T1

1 Objetivo

O objetivo desse trabalho prático é aprofundar a compreensão de comandos condicionais, comandos de repetição, vetores, matrizes, funções, registros/structs, arquivo binário, arquivo texto e ponteiros/alocação dinâmica de memória. Você **deve necessariamente e somente** utilizar os protótipos e códigos presentes nos arquivos *.c* e *.h* disponibilizados com esta especificação. Para tal, você implementará um sistema para gerenciar informações para um conjunto de imagens (formato PGM - portable graymap) bem como realizar algumas operações sobre as imagens armazenadas.

2 Descrição

O sistema é composto por duas partes: **1) manipulação de cadastro** com informações (denominado aqui de base) sobre imagens *.pgm*. Nesta parte do trabalho, serão implementadas funções para **listar, cadastrar, alterar e remover**[1] informações sobre as imagens na base. Funções para auxiliar a codificação das funções anteriormente citadas também serão implementadas. Estas informações serão manipuladas no arquivo binário *arqFisicoImagensBase.bin*. **2) A segunda parte do trabalho consiste na realização de operações** (**binarização, ruído, negativo, espelhamento**)[2] sobre as imagens (*pgm*) cadastradas. As imagens são manipuladas através de arquivo texto (exemplo: *lena.pgm*).

3 Códigos-fonte utilizados

As informações sobre uma imagem estão definidas na struct Imagem (ver "libTrabalho.h" disponibilizada com o código). Os cabeçalhos das funções necessárias para a implementação do trabalho (e descritas a seguir) estão, também, definidos em "libTrabalho.h". No código disponibilizado há, também, comentários para auxiliar na implementação do trabalho.

3.1 Funções necessárias para a parte de manipulação de cadastro de imagens (parte 1 do trabalho).

A função *menuOperacoes* (já implementada) lista as operações que podem ser realizadas no sistema. Protótipo:

```
void menuOperacoes();
```

[1] Detalhes na Seção 3.1.
[2] Detalhes na Seção 3.2.

A função *listaImagens* deve imprimir as informações de todas as imagens cadastradas na base (o parâmetro $arqFisicoImagensBase$ contém o nome da base, ou seja, $arqFisicoImagensBase.bin$). Protótipo:

```
void listaImagens(char arqFisicoImagensBase[]);
```

A função *cadastraImagem* deve adicionar uma imagem à base de cadastros $arqFisicoImagensBase$ de imagens. Esta função retorna zero se a imagem foi cadastrada com sucesso; retorna 1 se o arquivo já está cadastrado; retorna -1 nas demais situações. A informação retornada deve ser usada para emitir mensagem de sucesso/ou não no cadastramento. Protótipo:

```
int cadastraImagem(char arqFisicoImagensBase[], char nomeImagem[]);
```

Conforme especificado em $libTrabalho.h$, uma imagem é definida por:

```
//Registro para a definição do tipo Imagem
typedef struct imagem{
   char nome[40];
   int id;
   //Neste trabalho, o tipo considerado é PGM
   char tipo[7];
   int size;
   Dimension dimensao;
   char proprietario[MAX_NAME];
   Data data_criacao;
   Data data_modificacao;
} Imagem;
```

Note que uma nova imagem:

- só pode ser cadastrada com um nome diferente dos nomes existentes no arquivo $arqFisicoImagensBase$. Para verificar se o nome ($nomeImagem$) já existe na base a função $verificaCadastro$[3] deve ser implementada/usada.
- só pode ser cadastrada se existe uma imagem com o nome especificado no diretório corrente, ou seja, para cadastrar informações sobre uma imagem deve-se garantir que ela exista. Para fazer esta verificação use a função $verificaExistenciaDeImagem$ (já implementada).

Cada imagem cadastrada recebe um *id* (campo *id* na struct Imagem) que é subsequente ao *id* anteriormente cadastrado na base (ou seja, a primeira imagem cadastrada tem $id = 1$, a segunda $id = 2$,..). Para descobrir o último *id* cadastrado deve-se implementar/usar a função *getLastId*.

O campo *tipo* (na struct Imagem) é a extensão da imagem (que deve necessariamente ser *.pgm* neste trabalho), a qual deve ser capturada do nome da imagem a ser cadastrada ($nomeImagem$) através da função *getExtension* (implementar).

O tamanho da imagem (campo *size* na struct Imagem) deve ser capturado da imagem a ser cadastrada (e que está presente no diretório corrente) através da função *getSize* (já implementada).

A dimensão da imagem (campo *dimensao* na struct Imagem) deve ser capturada da imagem a ser cadastrada (e que está presente no diretório corrente) através da função *getDimension* a ser implementada.

[3]Esta e demais funções citadas (e que precisam ser implementadas) são descritas na sequência desta especificação.

A data de criação e de alteração de uma imagem (campos *data_criacao* e *data_modificacao* na struct Imagem) devem ser capturadas da data do sistema. Usar a função *getSystemTime* (já implementada) para esta captura. No arquivo *cadastraImagem.c* é descrito como as informações retornadas por *getSystemTime* devem ser utilizadas.

A função *verificaCadastro* é definida por:

```
int verificaCadastro(char arqFisicoImagensBase[], char nomeImagem[]);
```

Esta função deve verificar se no arquivo *arqFisicoImagensBase* existe alguma imagem com o mesmo nome de *nomeImagem*. Seu retorno é 1 caso o nome já esteja cadastrado; retorna zero se não está cadastrado e -1 para outros erros (por exemplo: abertura de arquivo).

A função *getExtension* retorna em *extension* a extensão (neste trabalho *.pgm*, mas deve ser genérico para qualquer extensão) do nome da imagem armazenado em *nomeImagem*. Protótipo:

```
void getExtension(char nomeImagem[], char extension[])
```

Por exemplo: para *lena.pgm*, no retorno da função, *extension* deve conter *.pgm*.

A função *getDimension* é responsável por capturar a dimensão (altura e largura) da imagem:

```
int getDimension(char nomeImagem[], int *pAltura)
```

Leia na Seção 3.2 sobre o número de colunas e linhas da imagem. Esta função tem como retorno a largura da imagem. Além disso, através do ponteiro *pAltura* deve-se retornar a altura da imagem.

A função *getLastId* retorna o último *id* cadastrado em *arqFisicoImagensBase*; caso a base esteja vazia, retorna-se zero. Protótipo:

```
int getLastId(char arqFisicoImagensBase[]);
```

A função *gravaInfoImagem* é responsável por gravar as informações da nova imagem *img* (ver struct Imagem) no final do arquivo *arqFisicoImagensBase*. Protótipo:

```
int gravaInfoImagem(char arqFisicoImagensBase[],  Imagem img)
```

A função *alteraImagem* é responsável por alterar as informações da imagem com *id* na base *arqFisicoImagensBase*. Protótipo:

```
int alteraImagem(char arqFisicoImagensBase[], int id);
```

Note que só podemos alterar informações se a imagem estiver cadastrada. Esta função retorna 1 caso o *id* exista ou zero caso contrário. O valor retornado pela função (0 ou 1) deve ser usado para imprimir mensagem de existência ou não da imagem na base. As informações que podem ser alteradas são: *proprietario* e *data_modificacao* (esta deve ser atualizada com base no retorno da função *getSystemTime*). **Dica:** use o *fseek* para a atualização em *arqFisicoImagensBase*.

A função *removeImagem* (já implementada) deve remover (se existir) a imagem com *id* da base *arqFisicoImagensBase*. A imagem também deve ser removida do diretório corrente (por exemplo, se foi solicitada a remoção das informações sobre a imagem *lena.pgm* da base, então *lena.pgm* também deve ser removida do diretório corrente).

3.2 Funções necessárias para realizar operações sobre as imagens (parte 2 do trabalho)

O usuário pode realizar operações sobre suas imagens favoritas no formato PGM. Este formato tem duas variações, uma binária (o PGM "normal" ou raw) e outra textual (o PGM ASCII ou plain). Em ambos os casos, o arquivo deve conter um cabeçalho e a matriz correspondente à imagem. O exemplo a seguir mostra um arquivo PGM textual:

```
P2
5 4
34
10  18  21   0   8
 1   7   2   7   1
 0   9  34  23  12
16  18   6  15  19
```

A primeira linha do arquivo contém obrigatoriamente uma palavra-chave que determina o **tipo** da imagem, no caso desse trabalho ela deve ser SEMPRE P2 (arquivo PGM textual). A segunda linha contém dois números inteiros que indicam o **número de** *colunas* **e o número de** *linhas* da matriz, respectivamente. A terceira linha contém um número inteiro positivo maxval, que deve ser igual ao maior elemento da matriz. Na definição do formato PGM, maxval não pode ser maior que 65535. Para fins deste trabalho prático, entretanto, maxval é SEMPRE 255. Os demais números do arquivo são os elementos de uma matriz de inteiros com os tons de cinza de cada ponto da imagem. Cada tom de cinza é um número entre 0 e maxval (255), com 0 indicando "preto" e 255 indicando "branco".

Para visualização das imagens você pode utilizar o programa **display** no linux ou o programa **irfanview** do windows (http://www.irfanview.com/).

Neste trabalho, as operações possíveis sobre imagens no formato PGM são espelhar, binarizar, negativo e ruído.

- **Espelhar** uma imagem, tal como no exemplo da Figura 1.

Figura 1: Imagem de entrada e sua respectiva imagem **espelhada**.

- Para **binarizar** uma imagem (todo pixel maior que 127 vira 255 (branco) e todo pixel menor que 127 vira 0 (preto)), tal como no exemplo da Figura 2.

Figura 2: Imagem de entrada e sua respectiva imagem **binarizada**.

- Para realizar o **negativo** de uma imagem — todo pixel 255 vira 0, todo pixel 0 vira 255, um pixel x vira 255 - x — tal como no exemplo da Figura 3.

Figura 3: Imagem de entrada e sua respectiva imagem **negativa**.

- Para adicionar um **ruído** na imagem — o ruído não deve ultrapassar um valor de 100 e deve ser gerado pela função rand(), o pixel adicionado ao ruído não pode ser menor que 0 nem maior que 255 — tal como no exemplo da Figura 4.

Para as operações sobre as imagens são necessárias as funções descritas a seguir.

Função *alocaString*: aloca dinamicamente uma string (vetor de caracteres) do tamanho especificado como parâmetro e retorna um ponteiro para a string alocada. Protótipo:

```
char *alocaString (int size);
```

Função *leArquivoImagem*: lê o arquivo físico *nomeArqEntrada* (um ponteiro para char), recebido como argumento da função *main*, e retorna um ponteiro de ponteiro para a matriz (alocada dinamicamente dentro desta função) preenchida com os valores de tons de cinza para cada ponto da imagem. Esta função tem ainda os parâmetros *tipo* (um ponteiro para char), *lin*, *col* e *maxval* (estes

Figura 4: Imagem de entrada e sua respectiva imagem com **ruído**.

três últimos são ponteiros para inteiro), os quais referem-se às informações sobre a imagem, ou seja, a palavra-chave, linha, coluna, e maxval. Caso não seja possível fazer a leitura do arquivo imagem, esta função deve retornar o valor $NULL$, o qual deve ser utilizado no retorno da função para a verificação da continuidade da execução do programa. Em caso da não continuidade da execução do programa, a mensagem "Problema ao abrir imagem em *leArquivoImagem*. Execução finalizada." deve ser emitida. Protótipo:

```
int **leArquivoImagem (char *nomeArqEntrada, char *tipo,
                       int *lin, int *col, int *maxval);
```

Função *alocaMatrizImagem*: aloca dinamicamente uma matriz de inteiros *lin X col* (parâmetros para esta função e que foram obtidos da leitura do arquivo imagem) e retornar um ponteiro de ponteiro para inteiro (matriz) para a memória alocada. Protótipo:

```
int **alocaMatrizImagem (int lin, int col);
```

Função *desalocaMatrizImagem*: desaloca a matriz *mat* recebida como parâmetro. Tem ainda os parâmetros *lin* (inteiro) e *col* (inteiro).

```
void desalocaMatrizImagem(int **mat, int lin, int col)
```

Atenção: alocar memória dinamicamente quando necessário e desalocar assim que sua utilização for finalizada.

Função *copiaMatrizImagem*:

```
void copiaMatrizImagem (int **mat, int **matCopia, int lin, int col)
```

faz uma cópia da matriz *mat* (matriz original lida do arquivo) para uma matriz auxiliar *matCopia* (alocada dinamicamente antes da chamada de *copiaMatrizImagem*). A operação de espelhamento usará *copiaMatrizImagem* para não sobrescrever as informações da matriz original.

Função *espelhar*: realiza a operação de espelhamento sobre a imagem, ou seja, sobre os valores da matriz *mat*. Protótipo:

```
void espelhar (int **mat, int lin, int col);
```

Função *binarizar*: realiza a operação de binarização sobre a imagem, ou seja, sobre os valores da matriz *mat*. Protótipo:

```
void binarizar (int **mat, int lin, int col);
```

Função *negativo*: realiza a operação de negativo sobre a imagem, ou seja, sobre os valores da matriz *mat*. Protótipo:

```
void negativo (int **mat, int lin, int col);
```

Função *ruido*: realiza a operação de ruído sobre a imagem, ou seja, sobre os valores da matriz *mat*.

```
void ruido (int **mat, int lin, int col);
```

Função *gravaImagem*: esta função é usada após o retorno da função para cada operação e é responsável por gravar a imagem resultante (em *mat*) em um arquivo de saída (*nomeArqSaida*). *tipo* (ponteiro par char) representa a informação *P2* presente no arquivo da imagem de entrada; *lin* (inteiro) e *col* (inteiro) representam o número de linhas e colunas da matriz *mat* (e que foram lidos dos arquivo imagem de entrada); *maxval* representa o valor de maxval lido do arquivo imagem de entrada; *mat* (ponteiro de ponteiro para inteiro) é a matriz com a imagem resultante da operação previamente realizada. A função *gravaImagem* retorna 0 caso o arquivo tenha sido gravado com sucesso ou retorna 1 caso não tenha sido gravado. O valor retornado deve ser usado para imprimir uma mensagem sobre o sucesso da operação, por exemplo, "Operação de binarização realizada com sucesso". Protótipo:

```
int gravaImagem (char *nomeArqSaida, char *tipo, int lin,
                          int col, int maxval, int **mat);
```

Atenção 1: note que uma operação só pode ser realizada sobre uma imagem $auxNomeImagem$ (digitado pelo usuário) se ela existe no diretório corrente (e, consequentemente, na base de informações).

Atenção 2: o usuário digita um nome (*nomeArqSaida*) para a nova imagem (a imagem resultante de uma operação é uma nova imagem). Logo, *nomeArqSaida* não pode existir no diretório correte (e, consequentemente, na base de informações) antes da operação. Você deve garantir que *nomeArqSaida* seja adequado (ou seja, digitar até ser adequado).

Atenção 3: como a imagem resultante é uma nova imagem, ela deve ser cadastrada na base ($arqFisicoImagensBase$). A única diferença da nova imagem para a imagem origem ($auxNomeImagem$) é o nome e data de criação. Assim, use a função *getImage* (já implementada) para pegar a imagem original $arqFisicoImagensBase$; altere as informações necessárias e grave a nova imagem na base (usando $gravaInfoImagem()$).

Os três itens mencionados no Atenção devem necessariamente ser implementados na função main (nos respecticos cases).

4 Imagens disponibilizadas

São disponibilizadas, junto com o código fonte, as imagens: *lena.pgm*, *baloes.pgm* e *barbara.pgm*. Você pode adicionar outras imagens à base.

5 Warnings

Na compilação inicial do trabalho você observará alguns ***warnings***, os quais são decorrentes da falta de inclusão da biblioteca utilizada (*libTrabalho.h*). A correta inclusão da biblioteca faz parte do trabalho.

6 Composição do trabalho

O trabalho, que pode ser realizado em dupla ou de forma individual, é composto pelas partes:

- Um diretório contendo código-fonte (contendo somente arquivos .c e arquivos .h) e Makefile (no estilo makefileSimple). **Obs: trabalhos com Makefile que não funcione ou diferente do estilo "makefileSimple" não serão corrigidos.** Devem ser utilizados necessariamente e **somente os arquivos .c e .h disponibilizados e com a devida implementação por você realizada**. **Obs: trabalhos com arquivos/diretórios de codeBlocks (ou outras IDEs ...) não serão corrigidos.**
- **Todos os códigos devem, necessariamente, estar indentados e conter comentários significativos**. A falta ou insuficiência de comentários e indentação acarretará em desconto na nota.
- **Caso necessário, a professora poderá solicitar a apresentação do trabalho.**

7 Informações sobre a entrega

- **Data de entrega no moodle** (compactar (.zip) o diretório com códigos): 2 de Dezembro de 2022 (sexta-feira) 23:59.

8 Informações adicionais

- A biblioteca *string.h* pode ser utilizada neste trabalho.
- Cópias (com ou sem eventuais disfarces) receberão nota ZERO.
- Trabalho atrasados serão descontados em 50% da nota por dia de atraso.
- Os códigos devem, necessariamente, ser entregues via **Moodle**. **Identifique o trabalho com o(s) número(s) de registro acadêmico: RAestudante1_RAestudante2.{zip}**.

9 Dicas

A partir do main, acompanhe a sequência de chamadas de funções para entender o fluxo de execução do programa.

Implemente primeiro a função *cadastraImagem* (e as funções necessários para sua implementação). Posteriormente, implemente a função *listaImagens*, pois ela permitirá que você verifique o resultado do cadastramento.

Escolha uma das operações (por exemplo binarizar) e implemente teste. O entendimento da implementação desta função se aplica para as demais operações.

Na hora de testar seu código, se você cadastrou ou alterou alguma informação de forma incorreta, **remova o arquivo** *arqFisicoImagensBase.bin* de forma a evitar o uso de informações incorretas.

Use printf para debugar o código. Por exemplo, você pode deixar printfs dentro função *cadastraImagem* indicando que aquele printf foi realizado dentro da função..., assim, você consegue mapear de onde vem um erro gerado. Obviamente, antes de entregar o trabalho deve-se comentar ou remover os printfs de debug.

Na dúvida, volte e releia a especificação de cada parte/função do trabalho.

Lembre de fechar arquivos abertos e desalocar memória alocada de forma dinâmica.

Lembre de abrir os arquivos de forma adequada. Reveja os modos de abertura (leitura, escrita, append, ...) de arquivo.