Centro de Matemática, Computação e Cognição – UFABC
MCTA028-15 – Programação Estruturada – 3º Quadrimestre de 2023
http://professor.ufabc.edu.br/~m.sambinelli/courses/2023Q3-PE/index.html

Laboratório 7: arquivos

Atenção: de agora em diante está terminantemente **proibido** usar arrays estáticos de tamanho variado como os exibidos abaixo:

```
int n;
int vetor[n]; // nunca definir a dimensão de um array com uma variável
```

Essa proibição aplica-se a tudo: listas, trabalho e avaliação. Qualquer programa usando o tipo de construção acima receberá nota **zero**. Todo array dinâmico deverá ser construído usando alocação dinâmica de memória:

```
int* vetor = calloc(n, sizeof(int)); // ou
int* vetor2 = malloc(n * sizeof(int));
```

Instruções

- Em todos os seus programas você deve gerenciar corretamente a memória, liberando toda a memória requerida pelo seu programa após o término do uso. Programas com vazamento de memória receberão uma penalização de 25% do valor da nota total do exercício. Você pode verificar se o seu programa possui vazamento de memória com o comando

  ```
  valgrind --leak-check=full /caminho/para/o/seu/programa
  ```

  Para que o comando acima funcione, você deve habilitar a *flag* de *debug* do seu compilador. O exemplo a seguir ilustra como deve ser feito caso você use o *gcc* para compilar o seu programa

  ```
  gcc -Wall -Wextra -Wvla -g -std=c99 arquivo.c
  ```

- Em vários exercícios, eu peço a vocês para que escrevam uma função de um determinado tipo. Além de escrever essa função, vocês também devem escrever uma função `main()` que irá usar essa função com dados fornecidos pelo usuário. Ou seja, a sua `main()` deverá pedir a entrada para o usuário e passar esses dados como parâmetro para a função que você desenvolveu. Requisite esses dados imprimindo mensagens na tela, para que o professor saiba o que digitar quando estiver corrigindo o seu trabalho.

Questão 1. Escreva o programa `wordcount`. Este programa recebe como argumento uma arquivo e imprime um único inteiro que representa o número de palavras contidas no arquivo.

Questão 2. Escreva o programa `meucat` que se comporta como o programa `cat` do Unix[1]. Esse programa recebe como entrada uma sequência de um ou mais arquivos de texto e imprime o conteúdo desses arquivos na tela do usuário seguindo a ordem em que foi fornecido. A seguir, apresento uma forma válida de invocar o seu programa:

[1] Unix é o sistema operacional avô de todo os sistemas operacionais bons: Linux, Mac Os, FreeBSD, OpenBSD, Solaris, Android

```
meucat arq01.txt arq02.txt senhas.csv ../alunos_pe.txt ../../alunos_pe2.txt
```

Obs: o seu programa deve funcionar para qualquer quantidade de arquivos fornecidos pelo usuário. Caso o usuário não forneça nenhum arquivo, ou ele forneça um arquivo inexistente, o seu programa deve fornecer uma mensagem de erro para o usuário.

Questão 3. Escreva o programa `meugrep` que se comporta como o programa `grep` do Unix. O programa `grep` faz a busca de um padrão em um texto. A nossa versão será bem mais humilde: ao invés do padrão ser descrito por uma expressão regular, descreveremos o padrão simplesmente escrevendo a palavra que estamos buscando.

O seu programa `meugrep` deve funcionar assim:

```
meugrep palavra arq01.txt arq02.txt arq03.txt arq04.txt (pode ter mais arquivos)
```

O primeiro argumento que o seu programa irá receber é uma palavra, esse será o padrão que o seu programa deverá buscar. Na sequência, o usuário fornecerá uma lista com o caminho para um ou mais arquivos de texto. O seu programa dever procurar pela palavra fornecida em cada um dos arquivos e, para cada ocorrência encontrada, o seu programa dever imprimir uma linha seguindo o seguinte padrão:

```
nome do arquivo.txt:80: esta eh a linha que contem a **palavra** dentro do arquivo
```

Na linha acima podemos ver que o seu programa deve imprimir o nome do arquivo no qual a palavra foi encontrada. O número que aparece na sequência, no caso do exemplo, 80, é o número da linha dentro do arquivo 'nome do arquivo.txt' que contém a palavra buscada. Após o número, o seu programa deve imprimir o conteúdo da linha 80, que contém a palavra buscada. Além disso, o seu programa deve destacar essa palavra dentro da linha colocando ** ao redor da palavra.

Ideia uma outra forma aceitável de destacar a palavra é fazer a impressão da palavra usando cores. Para isso, basta imprimir um código especial para definir a cor dos caracteres. O exemplo a seguir imprimir a palavra 'esperando' em vermelho:

```c
printf("Nao espere ");
printf("\033[1;31m"); // faz a impressão mudar para vermelho
printf("esperando");
printf("\033[1;m"); // reseta para a cor default
printf(". Espere vivendo!");
```

Para descobrir o código de outras cores e como escrever com underline, veja o link: https://www.shellhacks.com/bash-colors/

Questão 4. O formato PGM (*Portable Gray Map*) é um tipo de formato de arquivo de imagem usado para armazenar imagens em tons de cinza. Ele faz parte da família de formatos de imagem do Netpbm e é projetado para ser fácil de entender e processar. Abaixo estão algumas características principais do formato PGM:

1. **Tipo de Arquivo:**
   - Existem dois tipos principais de arquivos PGM: P2 e P5.

- O formato P2 é um formato ASCII, enquanto o P5 é um formato binário.

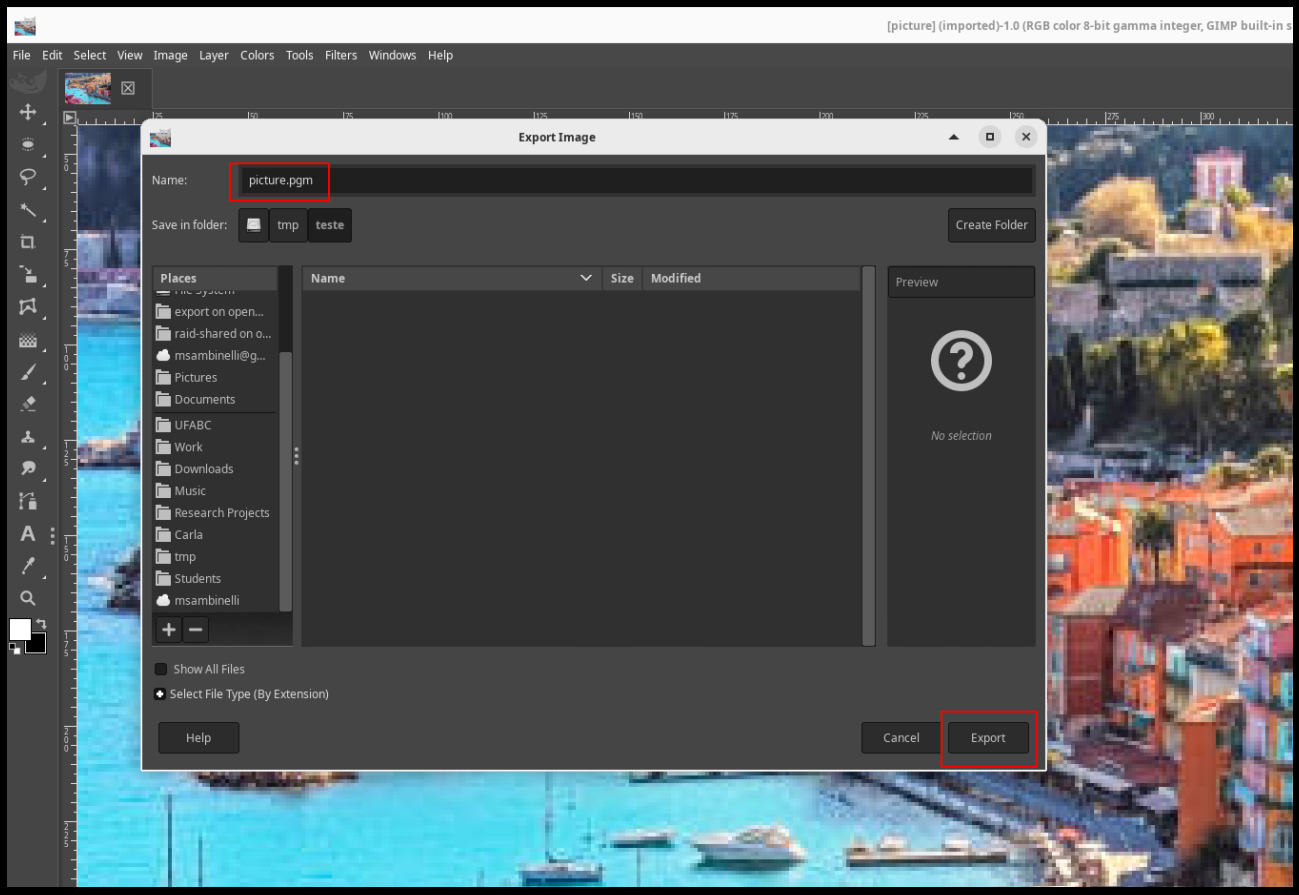
2. **Cabeçalho:**
   - Ambos os tipos começam com um cabeçalho que contém informações sobre a imagem, como largura, altura e o valor máximo do nível de cinza.
   - O cabeçalho começa com a mágica palavra ”P2”ou ”P5”, seguida por informações de largura e altura da imagem.
3. **Valores de Pixel:**
   - Após o cabeçalho, os valores de pixel são fornecidos.
   - No formato P2 (ASCII), os valores de pixel são listados como números inteiros separados por espaços ou quebras de linha.
   - No formato P5 (binário), os valores de pixel são armazenados como bytes.
4. **Representação de Nível de Cinza:**
   - Em um arquivo PGM, os valores de pixel representam os níveis de cinza da imagem.
   - O valor máximo (geralmente 255) indica o branco, enquanto zero representa o preto.
5. **Exemplo (P2):**

```
P2
# Comentário
4 4
255
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
```

No formato P5, a figura acima não teria o comentário e todos os dados seriam escrito em binário.

É muito fácil converter uma imagem colorida de um formato famoso, como png ou jpeg, para o formato pgm: basta um clique de botões, como a figura abaixo mostra[2].

[2] o software utilizado chama-se Gimp e é gratuito



Para processar uma imagem no formato pgm, lemos o arquivo texto (ou binário) que contém os valores dos pixels e carregamos esses valores em uma matriz. Assim, o processamento de uma imagem nada mais é que o processamento de uma matriz. Neste exercício, você irá ser capaz de aprender a fazer um filtro de borrão e um de detecção de borda. Assim, você será capaz de criar as seguintes imagens:

http://professor.ufabc.edu.br/~m.sambinelli/courses/2023Q3-PE/static/pictures.zip

Em processamento de imagens e visão computacional, um *kernel* refere-se a uma matriz (também conhecida como máscara ou filtro) que é usada para realizar operações como convolução em uma imagem. A convolução é uma técnica matemática que combina duas funções para produzir uma terceira, representando a maneira como uma função modifica a forma de outra.

Na aplicação específica de processamento de imagem, a convolução com um *kernel* é um processo fundamental para diversas operações, como filtragem, detecção de bordas e realce de características. Aqui está uma explicação básica da convolução de matriz:

(a) **Kernel:**
- Um *kernel* é uma matriz bidimensional de valores que é aplicada a uma matriz de pixels da imagem.
- Os valores no *kernel* determinam como a vizinhança de um pixel na imagem afeta o resultado da convolução.
- Exemplos de *kernels* incluem o *kernel* de identidade, o *kernel* de desfoque, o *kernel* de detecção de bordas, entre outros.

(b) **Convolução:**
- O processo de convolução envolve deslizar o *kernel* sobre a matriz de pixels da imagem, multiplicando os elementos do *kernel* pelos elementos correspondentes na vizinhança do pixel e somando os resultados.

- O resultado da soma é atribuído ao pixel central na nova imagem, criando uma nova matriz chamada de imagem convoluída.
- Este processo é repetido para cada pixel na imagem original.

A fórmula matemática para a convolução de matriz (denotada por I para a imagem e K para o *kernel*) em um ponto específico (i, j) é dada por:

$$(I * K)[i, j] = \sum_m \sum_n I[i - m][j - n] \cdot K[m, n]$$

Essa operação é central para muitas técnicas de processamento de imagem, como detecção de bordas, suavização, realce e outros filtros. *Kernels* diferentes produzem efeitos diferentes, permitindo a aplicação de várias transformações à imagem original.

O kernel para borrão é dado pela matriz

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

e o kernel para detecção de borda é dado pela matriz:

0	1	0
1	-4	1
0	1	0

Você deve implementar o programa `meuphotoshop` que recebe três parâmetros:

- uma palavra que pode ser ”borrao”ou ”borda”.
- um arquivo contendo uma figura pgm
- um arquivo que será gravado uma nova figura pgm

O seu programa deve ler a primeira figura fornecida e aplica o filtro especificado no primeiro parâmetro a figura e gravar o resultado no terceiro argumento.