FotoXop

Exercício Programa 1

Estrutura de Dados I

Prof. Oberlan C. Romão

Universidade Federal do Espírito Santo

Data de Entrega: até às 6h do dia 07/02/2026

Atenção: Este trabalho é **individual** e deve ser desenvolvido sem auxílio de ferramentas de inteligência artificial, geradores de código automáticos ou código de terceiros. A originalidade será verificada através de análise detalhada do código e entrevistas individuais. Casos de plágio ou uso de ferramentas não autorizadas resultarão em nota zero.

Sumário

1 Introdução

1.1 Sobre o Exercício Programa

Este Exercício Programa (EP) propõe o desenvolvimento de um sistema completo de manipulação de imagens digitais no formato PPM (*Portable Pixmap*), combinando conceitos fundamentais de estruturas de dados com processamento de imagens.

O trabalho está organizado em etapas progressivas que abordam:

- Manipulação básica de imagens (alocação, liberação e cópia)
- Aplicação de filtros e efeitos visuais
- Implementação de pilha para operações de desfazer
- Implementação de lista simplesmente encadeada
- Implementação de árvore AVL para indexação de cores
- Sistema de busca e substituição de cores

1.2 Conceitos de Imagens Digitais

Uma imagem digital é essencialmente uma matriz com $altura$ (número de linhas) e $largura$ (número de colunas). Cada elemento desta matriz é chamado de **pixel** (*picture element*), que possui uma informação de cor.

1.2.1 Tipos de Imagens

Imagens Binárias: Representam pixels como aceso ("branco") ou apagado ("preto"), necessitando apenas 1 bit por pixel.

Imagens em Tons de Cinza: Utilizam um byte (8 bits) por pixel, permitindo representar até 256 níveis de cinza.

Imagens Coloridas: Requerem mais informação por pixel. A representação mais comum utiliza o modelo RGB (*Red, Green, Blue*), decompondo cada cor em três componentes primárias. Cada componente é representada por um valor de 0 a 255.

1.2.2 O Formato PPM

O formato PPM (Portable Pixmap) é um formato bitmap simples, sem compressão, que armazena os valores de cada pixel sequencialmente. Nesse formato, as cores dos pixels são gravadas como números de 0 a 255. A primeira linha contém o tipo da imagem. No EP, usaremos apenas o tipo `P3` (imagem colorida ASCII). A segunda linha (e todas as seguintes que começarem com "`#`") contém algum comentário e podem ser descartadas. A terceira linha contém três valores, que indicam a dimensão da imagem, ou seja, largura e altura em pixels, e o valor máximo do pixel (255). Em seguida virão vários números, todos de 0 a 255, que indicam a cor de cada pixel. A quantidade depende do tamanho da imagem.

Considerando que a imagem é colorida, haverá pixels, cada um deles representado por três valores, todos entre 0 (min) e 255 (max). Esses 3 valores representam respectivamente a intensidade de Vermelho, Verde e Azul do pixel (padrão RGB). Veja o exemplo da figura abaixo: `255 0 0` indica `R=255, G=0, B=0`, ou seja, vermelho puro; `0 0 0` indica preto, `255 255 255` branco, `255 255 0` indica amarelo, etc. Na figura, a parte da esquerda é o resultado visto ao abrir o arquivo em um visualizador de imagens (GIMP, por exemplo). À esquerda é mostrado o conteúdo ao abrirmos a imagem em um editor de texto (VSCode, por exemplo).

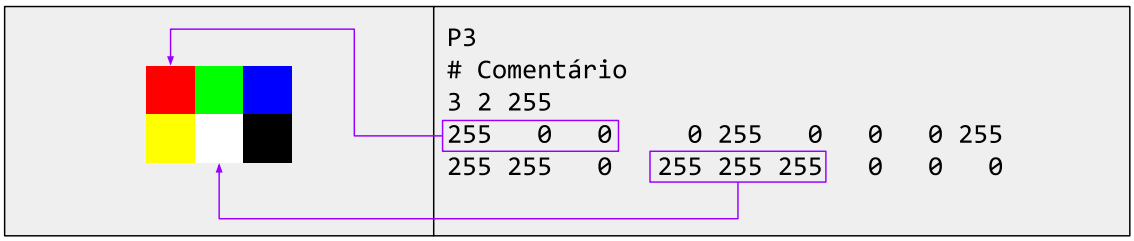


Figura 1.1: Exemplo de visualização de arquivo PPM: à esquerda a imagem renderizada, à direita o conteúdo do arquivo texto

Resumo: Estrutura do arquivo PPM

1. **Primeira linha:** Tipo da imagem (`P3`)
2. **Linhas de comentário:** Começam com "`#`" e podem ser ignoradas
3. **Linha de dimensões:** Contém largura, altura e valor máximo (255)
4. **Dados dos pixels:** Sequência de valores RGB (0-255) para cada pixel

1.3 Requisitos e Ferramentas

1.3.1 Biblioteca GTK3

O projeto utiliza a biblioteca GTK3 para criar uma interface gráfica interativa. Instruções de instalação específicas para cada sistema operacional estão disponíveis no AVA.

1.3.2 Editor de Imagens GIMP

O GIMP é recomendado para:

- Criar e exportar imagens no formato PPM
- Visualizar e editar imagens de teste
- Comparar resultados dos filtros implementados

1.3.3 Ambiente de Desenvolvimento

- **Linguagem**: C
- **Sistema Operacional:** Preferencialmente Linux
- **Editor:** VSCode (configuração detalhada no Capítulo 6)
- **Compilação:** Makefile fornecido

2 Estrutura do Projeto

2.1 Organização dos Arquivos

O projeto está dividido em módulos, cada um responsável por uma funcionalidade específica:

`AVL.h/c`	Implementação da árvore AVL para indexação de cores;
`Cor.h/c`	Funções auxiliares para manipulação de cores;
`Filtros.h/c`	Implementação dos filtros e efeitos visuais;
`Imagem.h/c`	Estruturas e funções para manipulação básica de imagens;
`Lista.h/c`	Implementação da lista simplesmente encadeada;
`main.c`	Arquivo principal (não deve ser modificado);
`Pilha.h/c`	Implementação da pilha para desfazer operações;
`Testes.h/c`	Funções responsáveis por realizar testes e validar as implementações de cada etapa;
`UI.h/c`	Interface gráfica e interação com usuário;
`Util.h/c`	Definições gerais e constante de controle da etapa.

Outros arquivos:

`boy.ppm / RGB.ppm`	Imagens de exemplos no formato PPM (no relatório utilize outras imagens);
`estilo.css`	Folha de estilo utilizado pelo GTK (se quiser, pode alterar para ter um programa com visual diferente);
`Makefile`	Arquivo responsável por limpar, compilar e executar o projeto;
`gtk.supp`	Indica quais funções devem ser ignoradas pelo `fsanitize`. Desta forma, não serão exibidos possíveis vazamentos de memória da implementação da biblioteca do GTK.

2.2 Sistema de Etapas

O desenvolvimento do EP está organizado em etapas incrementais. No arquivo `Util.h`, a constante `ETAPA` controla qual funcionalidade está ativa.

Atenção

Sempre que avançar para uma nova etapa:

1. Altere o valor de `ETAPA` em `Util.h`;
2. No terminal, execute os comandos `make clean && make run` para limpar, compilar e executar o projeto;
3. Teste completamente o programa antes de prosseguir;

Não pule as etapas. Cada etapa depende da implementação correta das anteriores.

2.3 Compilação e Execução

Comandos básicos:

- `make` – Compila o projeto
- `make run` – Compila e executa
- `make clean` – Remove arquivos objeto
- `make clean && make run` – Recompila tudo e executa

3 Etapas de Implementação

3.1 Etapa 0: Manipulação Básica de Imagens

3.1.1 Objetivo

Implementar as funções fundamentais para trabalhar com imagens: alocação de memória, liberação e cópia.

3.1.2 Funções a Implementar (arquivo `Imagem.c`)

`alocaImagem` Aloca dinamicamente a memória necessária para armazenar uma imagem com as dimensões especificadas

`liberaImagem` Libera toda a memória alocada para uma imagem, evitando vazamentos

`copiaImagem` Cria uma cópia completa de uma imagem existente

3.1.3 Validação

Após implementar as funções, compile e execute com `make clean && make run`. Se correto, a janela principal será exibida, como mostrado abaixo:

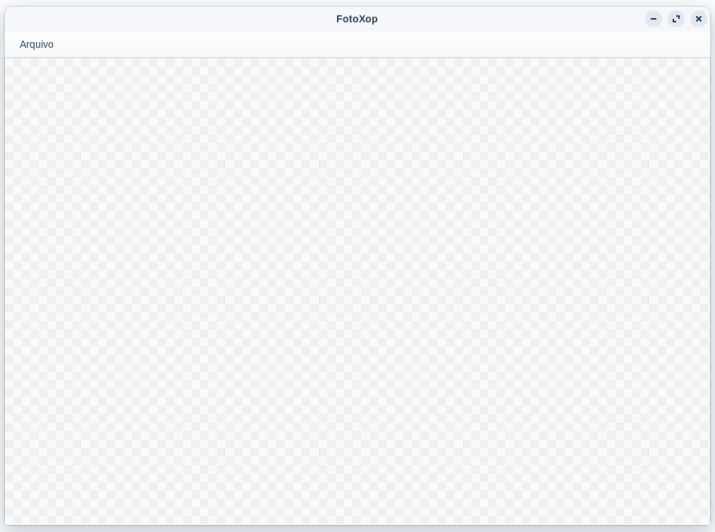


Figura 3.1: Janela principal do programa após completar a Etapa 0

3.1.4 Testando

Use o menu `Arquivo` para abrir uma imagem no formato PPM. Veja um exemplo:

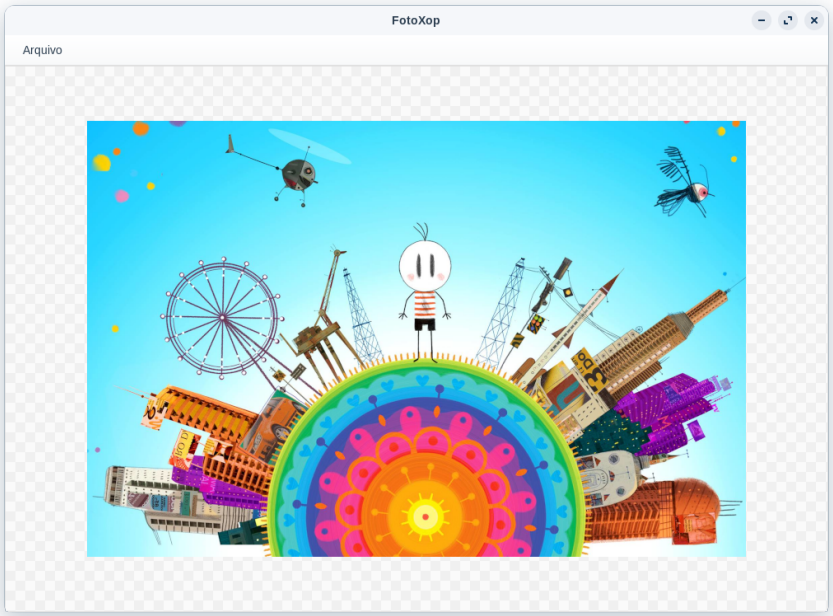


Figura 3.2: Exemplo de imagem carregada

3.2 Etapa 1: Aplicando Filtros e Efeitos

3.2.1 Objetivo

Implementar diversos filtros de processamento de imagens que modificam as cores e destacam características visuais.

3.2.2 Filtros a Implementar (arquivo `Filtros.c`)

Escala de Cinza

`escalaDeCinzaImagem`: Transforma uma imagem colorida em tons de cinza. Duas abordagens possíveis:

Opção 1 - Média simples:

$$\text{cinza} = \left\lceil \frac{R + G + B}{3} \right\rceil$$

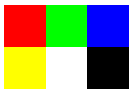
Imagem original	Escala de Cinza
P3 # Comentário 3 2 255 255 0 0 0 255 0 0 0 255 255 255 0 255 255 255 0 0 0 	P3 # Comentário 3 2 255 85 85 85 85 85 85 85 85 85 170 170 170 255 255 255 0 0 0 

Figura 3.3: Comparação: média simples

Opção 2 - Média ponderada:

$$\text{cinza} = \lceil 0.3 \times R + 0.59 \times G + 0.11 \times B \rceil$$

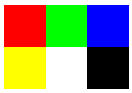
Imagem original	Escala de Cinza
P3 # Comentário 3 2 255 255 0 0 0 255 0 0 0 255 255 255 0 255 255 255 0 0 0 	P3 # Comentário 3 2 255 77 77 77 151 151 151 29 29 29 227 227 227 255 255 255 0 0 0 

Figura 3.4: Comparação: média ponderada (melhor resultado para cores primárias)

Filtro de Sobel

`filtroSobel`: Detecta bordas usando duas matrizes de convolução que identificam contornos verticais e horizontais. No EP você deve implementar uma versão desse filtro, mas que apresenta um excelente resultado. As matrizes utilizadas são

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Essas matrizes são aplicadas a cada banda de cor de cada pixel da imagem. Os coeficientes na matriz indicam o peso de cada pixel vizinho. No exemplo abaixo, a imagem à esquerda mostra uma parte da matriz de pixeis da imagem original. À direita é exibido como a matriz é aplicada a 3 desses pixeis.

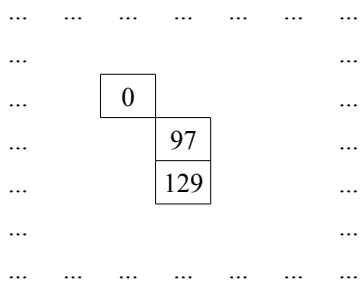
...	...	...	...	...	...	...
...	10	15	80	85	95	...
...	15	40	120	130	131	...
...	16	41	100	120	120	...
...	17	43	80	110	130	...
...	19	50	70	42	20	...
...	...	...	...	...	...	...

10*1 + 15*2 + 80*1 + 16*(-1) + 41*(-2) + 100*(-1) = -78

40*1 + 120*2 + 130*1 + 43*(-1) + 80*(-2) + 110*(-1) = 97

41*1 + 100*2 + 120*1 + 50*(-1) + 70*(-2) + 42*(-1) = 129

Para cada pixel, são utilizados seus 8 pixeis vizinhos. No caso da matriz os da linha de cima são somados (pesos 1 e 2), os da linha de baixo são subtraídos (pesos -1 e -2), e os da mesma linha não são considerados (peso 0). Abaixo o resultado para esses 3 pixeis. Os demais devem ser calculados da mesma forma.



Note que valores abaixo de 0 devem ser fixados em zero e valores acima de 255 devem ser fixados em 255. Se o valor resultante estiver entre 0 e 255, esse é o novo valor. Além disso, note também que o cálculo de cada pixel deve utilizar sempre os valores originais da imagem de entrada, e não os valores já modificados por cálculos de pixels vizinhos.

Aplicando essa matriz a todos os pixeis, a imagem resultante dará ênfase às bordas dos objetos da imagem (onde há mudanças brusca de cor).

> **Importante**
>
> **Você deve aplicar ambas as matrizes e combinar os resultados.** Escolha e implemente um dos métodos de combinação abaixo:
>
> - Soma dos valores absolutos
> - Média dos valores
> - Valor máximo entre G_x e G_y
> - Magnitude: $\sqrt{G_x^2 + G_y^2}$
>
> Independentemente do método escolhido, lembre-se que o valor final deve ser ajustado para o intervalo $[0, 255]$.

Recurso auxiliar: O site https://setosa.io/ev/image-kernels/ oferece uma visualização interativa de como as matrizes de convolução funcionam.

Detecção de Bordas de Laplace

`deteccaoBordasLaplace`: Similar ao Sobel, mas utiliza apenas uma matriz de convolução (apresentada abaixo) que deve ser aplicada em cada banda de cor de cada pixel.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Filtro Personalizado

`meuFiltro`: Implemente um filtro criativo de sua escolha. O filtro deve:

- Modificar efetivamente a imagem (pixel a pixel),
- Ser diferente dos filtros já implementados no EP,
- Funcionar corretamente para imagens coloridas (RGB),

Atenção – Requisitos

Filtros simples NÃO serão pontuados (nota 0):

- Negativo da imagem
- Espelhamento
- Escurecimento/clareamento simples
- Outras modificações triviais pixel a pixel

O filtro deve demonstrar criatividade e complexidade. Exemplos válidos:

- Filtros que usam vizinhança de pixels
- Efeitos artísticos elaborados
- Combinação de múltiplas técnicas
- Algoritmos de processamento avançados

No relatório, você deve explicar detalhadamente seu filtro.

3.2.3 Testando os Filtros

1. Altere `ETAPA` para 1 em `Util.h`
2. Compile: `make clean && make run`
3. Abra uma imagem e teste cada filtro através do menu `Filtros`

3.3 Etapa 2: Implementação de Pilha

3.3.1 Objetivo

Implementar uma pilha usando lista encadeada para permitir desfazer operações na imagem.

3.3.2 Funções a Implementar (arquivo `Pilha.c`)

A pilha deve seguir o princípio LIFO (*Last In, First Out*):

`criaPilha`	Inicializa uma nova pilha vazia

`liberaPilha`	Libera a memória alocada
`pushPilha`	Adiciona elemento no topo
`popPilha`	Remove elemento do topo
`topPilha`	Retorna elemento do topo

3.3.3 Validação

Altere `ETAPA` para 2 e compile: `make clean && make run`

3.4 Etapa 3: Integração da Pilha para Desfazer

3.4.1 Objetivo

Integrar a pilha ao programa principal, permitindo desfazer operações através da interface.

3.4.2 Funções a Implementar (arquivo `UI.c`)

Você deve completar as funções que:

- Salvam a imagem atual na pilha antes de aplicar um filtro/operação;
- Recuperam a última imagem da pilha ao desfazer;
- Gerenciam a memória adequadamente.

3.4.3 Funcionalidade

1. Ao aplicar filtro, a imagem original é automaticamente salva
2. Menu `Editar` → `Desfazer` recupera a última imagem
3. Possível desfazer múltiplas operações sequencialmente

3.4.4 Testando

1. Altere `ETAPA` para 3 em `Util.h`
2. Compile: `make clean && make run`
3. Abra uma imagem
4. Aplique diversos filtros
5. Use `Desfazer` para reverter as operações

3.5 Etapa 4: Lista Simplesmente Encadeada

3.5.1 Objetivo

Implementar lista simplesmente encadeada para armazenar posições (linha e coluna) onde determinada cor aparece.

3.5.2 Funções a Implementar (arquivo `Lista.c`)

`liberaLista`	Libera toda memória da lista
`insereLista`	Insere um novo elemento (posição) na lista
`appendLista`	Adiciona todos os elementos de uma lista no final de outra lista. A lista original deve ser preservada. Use `insereLista` para facilitar a implementação.

3.5.3 Validação

1. Altere `ETAPA` para 4 em `Util.h`
2. Compile e execute: `make clean && make run`

3.6 Etapa 5: Árvore AVL

3.6.1 Objetivo

Implementar uma árvore AVL para indexar as cores da imagem, permitindo busca eficiente de pixels por cor.

3.6.2 Estrutura

Cada nó contém:

- Uma cor (RGB)
- Uma lista encadeada com todas as posições onde essa cor ocorre
- Altura do nó
- Ponteiros para filhos esquerdo e direito

3.6.3 Funções a Implementar (arquivo `AVL.c`)

`criaArvore`	Cria uma árvore AVL a partir de uma imagem, indexando todas as cores e suas posições
`liberaNosArvore`	Libera recursivamente todos os nós da árvore, incluindo as listas associadas
`insereNo`	Insere um novo nó mantendo as propriedades da AVL: • Se a cor já existe, adiciona a posição à lista existente • Se é cor nova, cria novo nó com a cor e posição • Mantém o balanceamento através de rotações • Use `comparaCores` para ordenação
`buscaCorExata`	Busca a lista de posições para uma cor específica (tolerância = 0)
`buscaCorAproximada`	Busca todas as cores dentro de uma tolerância especificada:

- Percorre a árvore verificando distância entre cores
- Use `distanciaCores` para calcular similaridade
- Retorna lista agregada de todas as posições encontradas

`buscaArvore` Função principal que decide entre busca exata ou aproximada baseada na tolerância

> **Requisito Obrigatório**
>
> Esta etapa **exige o uso efetivo da Árvore AVL como estrutura principal** para indexação/busca de cores. Implementações que utilizem outra estrutura, ou que não utilizem a AVL de forma funcional, receberão **nota zero** nesta etapa.

3.6.4 Validação

1. Altere `ETAPA` para 5 em `Util.h`
2. Compile: `make clean && make run`
3. Teste com imagens de diferentes tamanhos

3.7 Etapa 6: Busca e Troca de Cores

3.7.1 Objetivo

Implementar funcionalidade completa de busca e substituição de cores usando a árvore AVL.

3.7.2 Função a Implementar (arquivo `UI.c`)

`previewTrocaCor` Implementa o fluxo completo:

1. Busca na árvore AVL as posições da cor selecionada
2. Aplica a tolerância especificada
3. Substitui todas as cores encontradas pela nova cor
4. Atualiza a visualização

3.7.3 Cálculo de Distância

A tolerância usa a distância Euclidiana no espaço RGB:

$$d = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

- Tolerância = 0: Apenas cor exata
- Tolerância > 0: Todas as cores com $d \leq$ tolerância

3.7.4 Interface de Uso

1. Altere `ETAPA` para 6 em `Util.h`
2. Compile: `make clean && make run`
3. Abra uma imagem
4. Clique com botão direito sobre um pixel
5. Selecione `Trocar cor` no menu de contexto
6. Configure na janela que abre (conforme imagem abaixo):
    - Nova cor RGB
    - Tolerância desejada
7. Teste com:
    - Tolerância 0 (cor exata)
    - Diferentes valores de tolerância
    - Imagens com variadas paletas de cores
8. Clique em `Aplicar`

Figura 3.5: Janela de troca de cores

4 Boas Práticas e Requisitos

4.1 Tratamento de Tipos de Dados

4.1.1 Cuidados com `Byte`

Valores do tipo `Byte` (`unsigned char`) estão limitados a $[0, 255]$. Operações podem causar *overflow*:

> ⚠ **Exemplos de Overflow**
>
> - $255 + 1 = 256 \longrightarrow$ armazenado como 0
> - $0 - 1 = -1 \longrightarrow$ armazenado como 255

Solução

1. Use `int` para cálculos intermediários
2. Antes de converter para `Byte`:
    - Se valor $> 255 \longrightarrow$ use 255
    - Se valor $< 0 \longrightarrow$ use 0
3. A função `ajustaCor` já faz esta conversão

4.2 Modularização e Organização

4.2.1 Estrutura de Código

- **Funções auxiliares:** Crie quando apropriado, adicionando-as nos arquivos `.c`
- **Evite duplicação:** Reutilize código através de funções
- **Tamanho de funções:** Mantenha funções concisas e com propósito único
- **Não altere assinaturas:** Mantenha nomes, parâmetros e tipos de retorno das funções fornecidas

4.3 Comentários e Documentação

Use comentários para explicar:

- Algoritmos complexos ou não triviais

- Decisões importantes de implementação
- Parâmetros de entrada e saída de funções auxiliares
- Trechos que possam gerar dúvidas

Não comente o óbvio! Comentários devem agregar valor.

4.4 Nomenclatura

Siga o Guia de Estilo disponibilizado no AVA:

- Variáveis: nomes descritivos em `camelCase`
- Funções: verbos que descrevem a ação

4.5 Gerenciamento de Memória

Atenção

- Toda memória alocada deve ser liberada
- Verifique retornos de `malloc` / `calloc`

Código com vazamento de memória valerá apenas **70%** da nota do EP.

4.6 Testes Recomendados

Antes de entregar, teste com:

4.6.1 Variedade de Imagens

- Pequenas (10×10, 50×50)
- Médias (500×500)
- Grandes (1000×1000 ou mais)

4.6.2 Características Diversas

- Muitas cores distintas
- Poucas cores (paleta limitada)
- Gradientes suaves
- Bordas nítidas

4.6.3 Sequências de Operações

- Aplicar múltiplos filtros consecutivos
- Desfazer várias vezes
- Trocar cores com diferentes tolerâncias
- Abrir e fechar múltiplas imagens

5 Entrega e Avaliação

5.1 O que Entregar

Entregue pelo **AVA** um arquivo `EP1.zip` contendo:

1. Todos os arquivos do projeto (não inclua arquivos objetos, ou seja, arquivos `.o`)
2. Relatório em PDF

Data da entrega: até às 6h do dia 07/02/2026

Lembre-se

Este trabalho é **individual** e deve ser desenvolvido sem auxílio de ferramentas de inteligência artificial, geradores de código automáticos ou código de terceiros. A originalidade será verificada através de análise detalhada do código e entrevistas individuais. **Casos de plágio ou uso de ferramentas não autorizadas resultarão em nota zero a todos os envolvidos.** Cuidado com implementações de alunos dos semestres anteriores, isso também é considerado plágio.

5.2 Requisitos do Código

- O código deve compilar e executar **sem erros** usando o comando `make run`;
- Todas as funções solicitadas devem estar implementadas;
- Não altere os nomes dos arquivos fornecidos nem a assinatura das funções (nome, parâmetros e tipo de retorno);
- Siga o Guia de Estilo disponibilizado no AVA;
- Inclua comentários explicativos em trechos complexos do código.

5.2.1 Testes Recomendados

Antes de entregar, teste seu programa com:

- Imagens de diferentes tamanhos (pequenas, médias e grandes);
- Imagens com diferentes características (muitas cores, poucas cores, gradientes);
- Sequências de operações: aplicar múltiplos filtros e desfazer várias vezes;
- Troca de cores com tolerância 0 e com diferentes valores de tolerância;

5.2.2 O que NÃO fazer

- **NÃO** altere a estrutura de diretórios fornecida;
- **NÃO** altere o `Makefile` (a menos que seja absolutamente necessário e justificado no Relatório);
- **NÃO** altere as assinaturas das funções fornecidas nos arquivos `.h`;
- **NÃO** inclua arquivos objeto (`.o`), executáveis ou arquivos temporários no arquivo compactado;
- **NÃO** inclua as imagens (arquivos PPM) no arquivo `.zip`;
- **NÃO** use bibliotecas externas além das já incluídas no projeto (GTK3 e bibliotecas padrão C);
- **NÃO** inclua mensagens de debug excessivas no código final.

> **Dica importante**
>
> Após criar o arquivo compactado, descompacte-o em um diretório temporário diferente e teste se tudo compila e executa corretamente. Isso garante que você não esqueceu de incluir algum arquivo necessário.

5.3 Requisitos do Relatório

O relatório deve ser entregue em formato PDF, em estilo acadêmico (normas ABNT ou equivalente) e conter as seguintes seções:

1. **Identificação:** Nome completo, matrícula e turma.
2. **Introdução:** Breve descrição do trabalho realizado (1-2 parágrafos).
3. **Implementação do Filtro Personalizado (`meuFiltro`):**
    - Descrição detalhada do filtro/efeito implementado;
    - Explicação do algoritmo utilizado (pode incluir pseudocódigo);
    - Se aplicável, mencione as referências consultadas.
4. **Resultados e Exemplos:**
    - Para cada filtro implementado (`escalaDeCinzaImagem`, `filtroSobel`, `deteccaoBordasLaplace` e `meuFiltro`), apresente:
        - Imagem original;
        - Imagem após aplicação do filtro;
        - Breve comentário sobre o resultado obtido.
    - Use pelo menos **3 imagens diferentes** para demonstrar os filtros;
    - As imagens devem estar em boa resolução e adequadamente legendadas.
5. **Funcionalidade de Troca de Cores:**
    - Apresente exemplos de uso da funcionalidade de busca e troca de cores;

- Mostre casos com tolerância 0 (cor exata) e com tolerância maior que 0;
- Inclua a imagem antes e depois da troca de cores.

6. **Estruturas de Dados:**
   - Explique brevemente como a Pilha foi utilizada para implementar a funcionalidade de desfazer operações;
   - Comente sobre a escolha da Árvore AVL para indexação de cores e suas vantagens em relação a outras estruturas.
7. **Conclusão:**
   - Considerações finais sobre o trabalho;
   - Aprendizados obtidos com o desenvolvimento do EP.

Observações sobre o relatório

- O relatório deve ter entre 8 e 15 páginas (incluindo as imagens);
- Use fonte legível (tamanho 11 ou 12) e margens adequadas;
- O texto deve estar justificado;
- Todas as imagens devem estar em boa qualidade e possuir legendas descritivas;
- Organize o relatório de forma clara e estruturada;
- Revise o texto para evitar erros ortográficos e gramaticais;
- **Não inclua** trechos de código no relatório, apenas explique a lógica implementada quando necessário;
- Relatórios muito curtos, sem exemplos adequados ou que não sigam a estrutura solicitada terão descontos na nota ou, em casos especificos, não serão considerados.

5.4 Checklist Antes de Enviar

☐ Código compila: `make run`

☐ Todas as 6 etapas implementadas e testadas

☐ Código segue o Guia de Estilo

☐ Arquivos desnecessários removidos

☐ Relatório em PDF incluído no arquivo `zip`

☐ Nome do arquivo: `EP1.zip`

☐ Testado após descompactar em diretório limpo

5.5 Critérios de Avaliação

A nota do EP se dará pela seguinte fórmula:

$$\text{Nota Final} = (1 - P) \times R \times M \times G \times (N_{EP} + N_R)$$

onde,

P (Plágio) $$P = \begin{cases} 1, & \text{se houve plágio;} \\ 0, & \text{caso contrário.} \end{cases}$$

R (Relatório) $$R = \begin{cases} 1, & \text{se entregou o relatório como solicitado;} \\ 0, & \text{caso contrário (ou se enviou um arquivo em branco).} \end{cases}$$

M (Memória) $$M = \begin{cases} 1.0, & \text{se o código não possui vazamento de memória;} \\ 0.7, & \text{caso contrário.} \end{cases}$$

G (Guia de Estilo) $$G = \begin{cases} 1.0, & \text{se seguiu o Guia de Estilo;} \\ 0.9, & \text{caso contrário.} \end{cases}$$

N_{EP} Nota da implementação: $0.0 \leq N_{EP} \leq 5.0$

N_R Nota do relatório (caprichem!): $0.0 \leq N_R \leq 5.0$

Ponto extra para os melhores trabalhos!

Atenção

- A avaliação considerará não apenas se o código funciona, mas também a qualidade da implementação, demonstração de compreensão dos conceitos das estruturas de dados utilizadas e a capacidade de documentar adequadamente o trabalho realizado.
- Penalidades como vazamento de memória e não seguir o Guia de Estilo reduzem a nota final multiplicativamente, mesmo que o programa funcione corretamente.

5.5.1 Penalidades

- Plágio ⟶ Nota 0
- Sem código ⟶ Nota 0
- Não compila ⟶ Nota 0
- Sem relatório ⟶ Multiplica por 0
- Vazamento de memória ⟶ Multiplica por 0.7
- Não seguir Guia ⟶ Multiplica por 0.9
- Etapa 5 sem AVL ⟶ Nota 0 nesta etapa

6 Configuração do Ambiente

6.1 Configurando o VSCode

O VSCode pode não reconhecer automaticamente as bibliotecas GTK3. Para resolver este problema, siga os passos abaixo:

- Abra o VSCode na pasta do EP;
- Pressione `Ctrl + Shift + P` e digite `C/C++: Edit Configurations (UI)`;
- No campo `Include path` (ou `Incluir caminho`), adicione os caminhos das bibliotecas do GTK3. No Linux, normalmente são esses caminhos:

```
/usr/include/gtk-3.0
/usr/include/at-spi2-atk/2.0
/usr/include/at-spi-2.0
/usr/include/dbus-1.0
/usr/lib/x86_64-linux-gnu/dbus-1.0/include
/usr/include/gtk-3.0
/usr/include/gio-unix-2.0
/usr/include/cairo
/usr/include/pango-1.0
/usr/include/harfbuzz
/usr/include/pango-1.0
/usr/include/fribidi
/usr/include/harfbuzz
/usr/include/atk-1.0
/usr/include/cairo
/usr/include/pixman-1
/usr/include/uuid
/usr/include/freetype2
/usr/include/gdk-pixbuf-2.0
/usr/include/libpng16
/usr/include/x86_64-linux-gnu
/usr/include/libmount
/usr/include/blkid
/usr/include/glib-2.0
/usr/lib/x86_64-linux-gnu/glib-2.0/include
```

- Salve o arquivo e reinicie o VSCode.

6.2 Criando Arquivos PPM com GIMP

6.2.1 Procedimento

1. Abra imagem no GIMP
2. Vá em: `Arquivo` → `Exportar como`
3. Altere extensão para `.ppm`
4. Clique em `Exportar`
5. Selecione formato `ASCII`
6. Clique em `Exportar`

6.2.2 Verificação

O arquivo deve começar com:

```
P3
# Comentário (opcional)
largura altura
255
R G B R G B ...
```

"Programming is not about typing,
it's about thinking."

— Rich Hickey

A Exemplos de Resultados Esperados

Esta seção apresenta exemplos visuais dos resultados esperados para cada filtro implementado.

A.1 Imagem Original

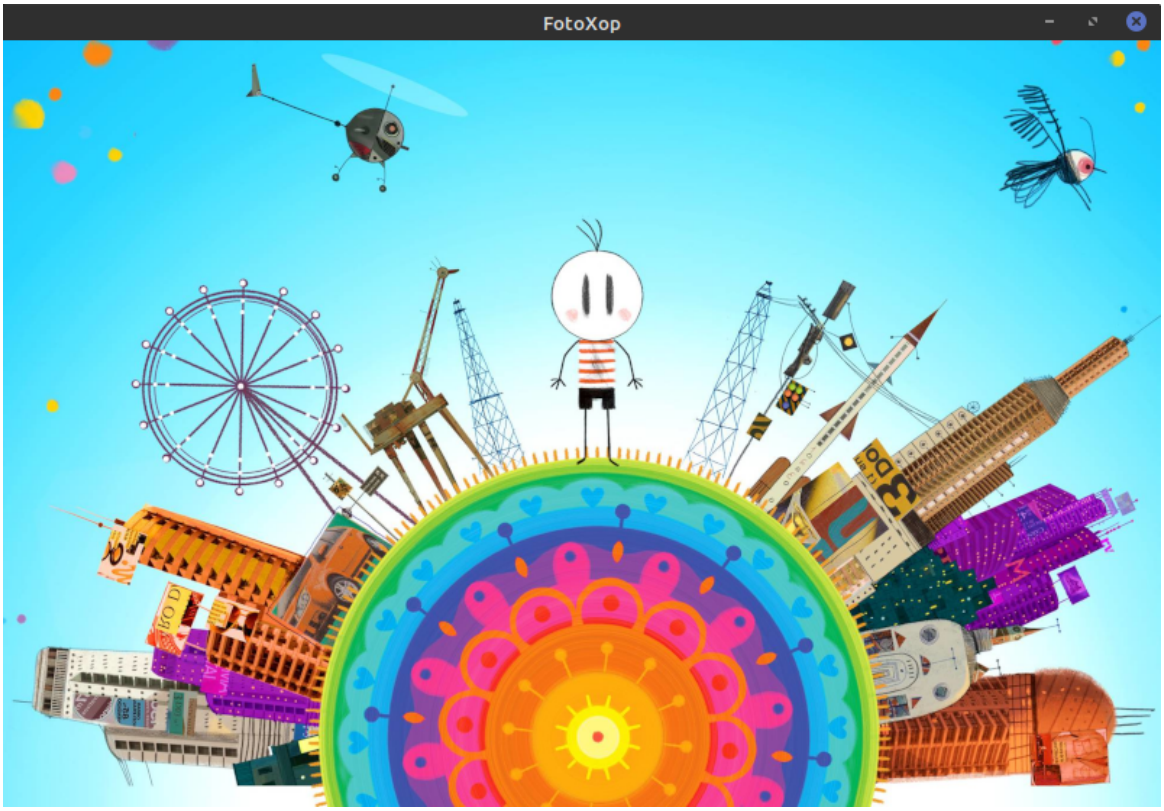


Figura A.1: Imagem original de referência para os exemplos

A.2 Escala de Cinza

Figura A.2: Resultado do filtro de escala de cinza

A.3 Filtro de Sobel

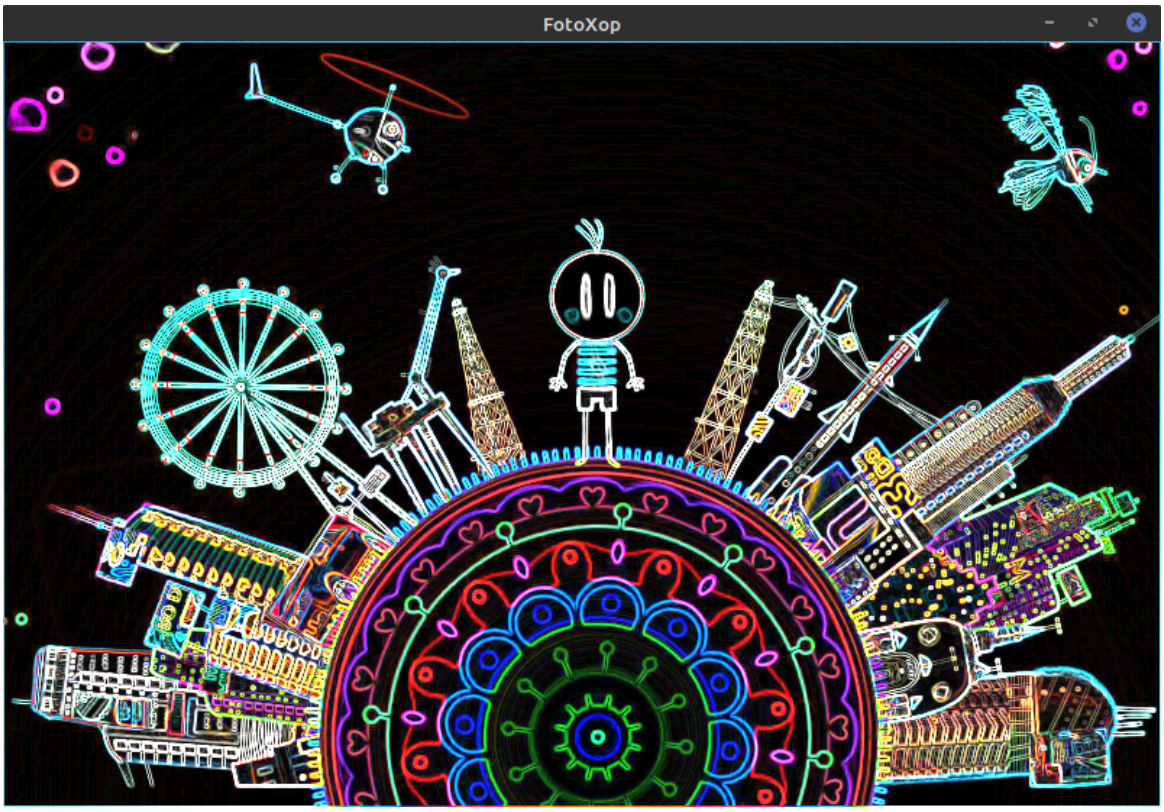


Figura A.3: Resultado do filtro de Sobel para detecção de bordas (o resultado pode variar dependendo do método de combinação escolhido)

A.4 Detecção de Bordas de Laplace

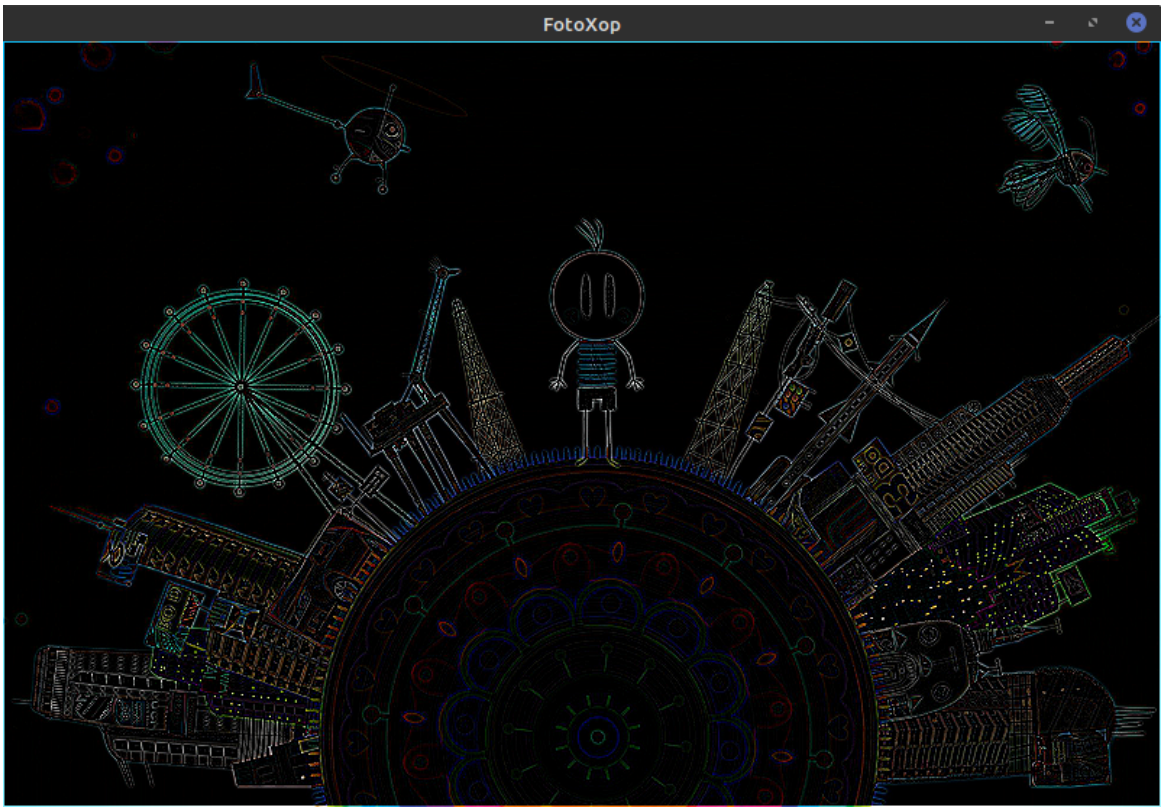


Figura A.4: Resultado da detecção de bordas usando o operador de Laplace