

Relatório Linguagens de Programação - Etapa 2

Ricardo Santos Siqueira

118167558

15 de outubro de 2021

1. Introdução

O presente trabalho implementa uma ferramenta para download, corte e processamento de imagens. Dessa forma o usuário pode de maneira simples entrar com a URL de uma imagem e realizar algumas operações disponibilizadas pelo programa na mesma de forma rápida e prática e salvar a imagem editada.

2. Descrição

O programa implementado consiste em uma CLI implementada em C++ que integra um script Python para download e manipulação de imagens. Dessa forma temos uma interface rápida e robusta para prover melhor usabilidade para o usuário, enquanto a implementação em Python é simples e rápida, por tratar-se de uma linguagem de alto nível e possuir bibliotecas que agilizam a implementação.

3. Motivação

A presente solução surgiu da necessidade de realizar pequenos ajustes de cor e contraste ou cortes em imagens. A ferramenta destaca-se pela funcionalidade que permite facilmente mapear o rosto de uma pessoa e ajustar a proporção da imagem para 3:4, formato padrão para preenchimento de diversos tipos de cadastros e documentos. Demanda que cresceu durante a pandemia, com a necessidade de submissão desse tipo de arquivos via internet.

4. Casos de uso

Uma vez iniciado o programa será solicitado ao usuário que entre com a URL da imagem que deseja interagir. Após a inserção de um valor válido ele será exposto à um menu poderá selecionar uma das seguintes opções, para interagir com a imagem:

- 1 - Cortar faces:** Executa o algoritmo de detecção de imagens para criar cópias dos rostos identificados.
- 2 - Ajustar a proporção da imagem para 3:4:** Ajusta a proporção das imagens no buffer para o formato 3:4.

3 - [Filtro] Suavizar detalhes: Suaviza os detalhes das imagens no buffer aplicando um efeito de gaussian blur.

4 - [Filtro] Realçar detalhes: Realça os detalhes das imagens no buffer.

5 - [Filtro] Ajustar cor: Ajusta a cor das imagens salvas no buffer de acordo com o valor fornecido pelo usuário.

6 - [Filtro] Ajustar contraste: Ajusta o contraste das imagens salvas no buffer de acordo com o valor fornecido pelo usuário.

7 - Salvar cópias e sair: Salva uma cópia de cada imagem no buffer e encerra a aplicação.

Além disso, todas as opções listadas acima, com exceção da opção 7, possuem submenus em que o usuário entra com os parâmetros para executar a ação ou pode retornar ao menu anterior.

5. Implementação

A implementação do programa pode ser dividida em duas partes principais, a CLI em C++ e o script Python, conforme diagrama abaixo:

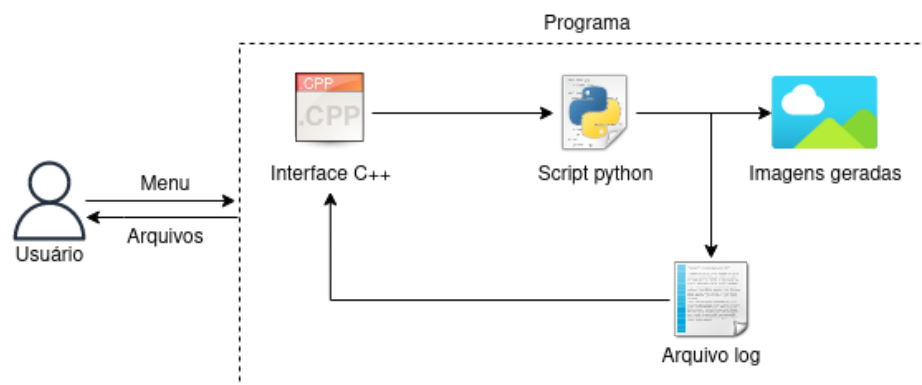


Figura 1: Diagrama da arquitetura da solução.

A interface entre as duas linguagens é feita via chamada de sistema e o arquivo texto log.tmp. Inicialmente foi cogitada a possibilidade da utilização da biblioteca `<Python.h>`, contudo as bibliotecas utilizadas no script Python não podiam ser importadas via esse método. Sendo assim, essa opção foi descartada e a implementação da interface foi feita via chamadas de sistema UNIX.

O comando enviado pela interface CLI em C++ e lido pelo script Python segue o seguinte formato:

```
python python/interface.py
# Espera uma string com uma url valida
<url da imagem :: string>
# Espera um booleano indicando se o usuario selecionou a opcao [1] do menu
<cortar faces :: boolean>
# Espera um booleano indicando se o usuario selecionou a opcao [2] do menu
<ajuste de proporcao :: boolean>
# Espera um booleano indicando se o usuario selecionou a opcao [3] do menu
<filtro de desfoque :: boolean>
# Espera um booleano indicando se o usuario selecionou a opcao [4] do menu
<filtro de detalhes :: boolean>
# Espera um float indicando a intensidade do ajuste, opcao [5] do menu
<ajuste de cor :: float = 1.f>
# Espera um float indicando a intensidade do ajuste, opcao [6] do menu
<ajuste de contraste :: float = 1.f>
# Operador de redirecionamento, readireciona o retorno do comando para o arquivo tmp.log
> tmp.log
```

Figura 2: Parâmetros do comando enviado pela CLI ao script.

a. CLI C++

A função principal do programa, **main.cpp**, implementa a classe **Menu** e envia o comando para o script Python via chamada de sistema. Finda a chamada de sistema, o retorno do script é lido no arquivo **tmp.log** e são indicados os nomes das imagens geradas pela execução do programa.

```
main.cpp

#include "Menu.hpp"
#include "FileResolver.hpp"

int main(int argc, char *argv[])
{
    const string logFile = "tmp.log";

    // Instanciando a classe Menu
    Menu menu = Menu();
    // Inicializando o menu interativo
    menu.run();

    // Menu finalizado

    // Chamada de sistema para o script python
    string command = "python python/interface.py " + menu.parseCommand() + " > " + logFile;

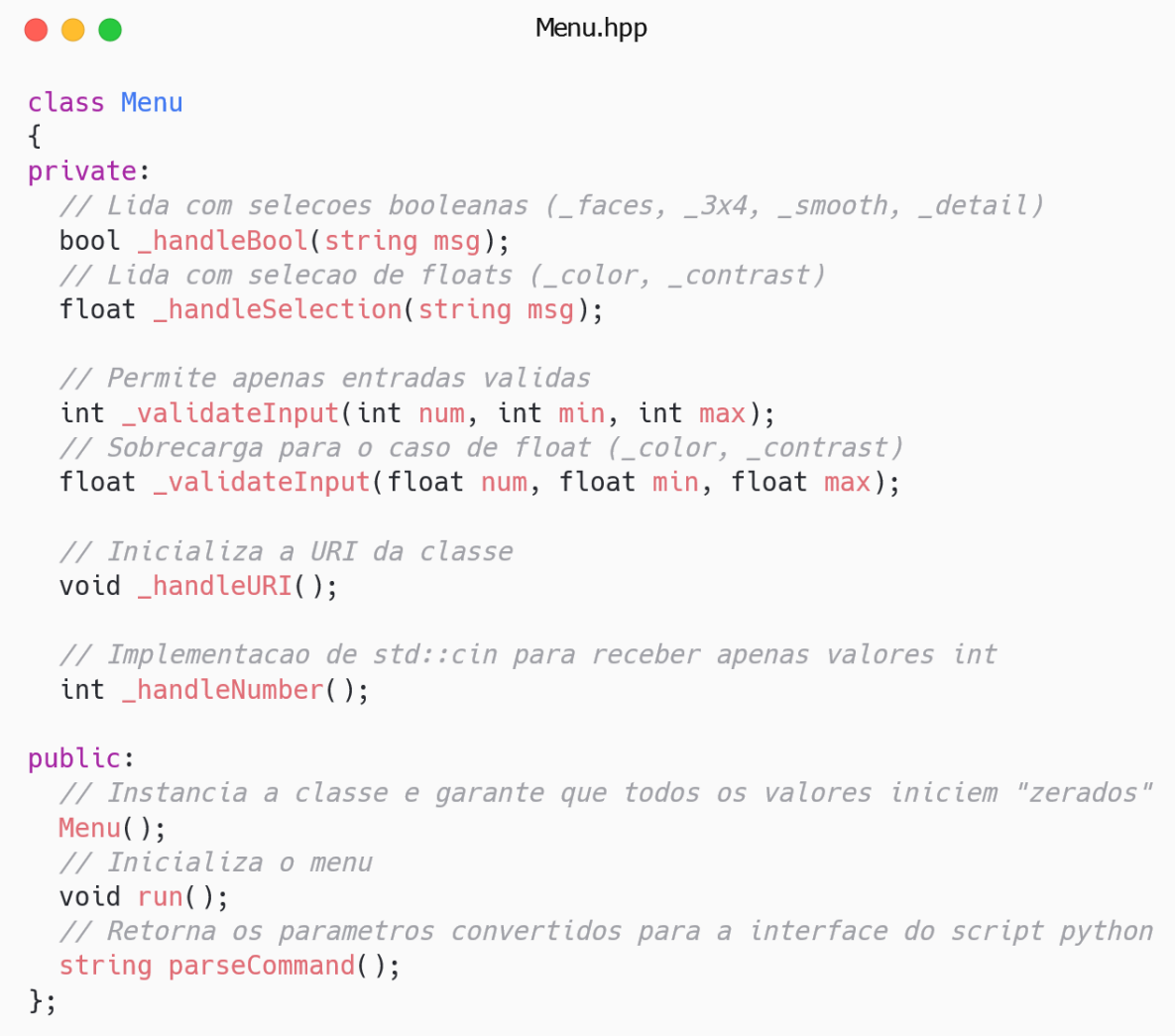
    system(command.c_str());

    // Lendo o arquivo log.txt
    FileResolver fileResolver(logFile);
    fileResolver.readFile();

    return 0;
}
```

Figura 3: Fragmento da implementação da função principal da CLI.

A classe “**Menu.hpp**” implementa dois métodos públicos, **Menu::run()** inicia o loop do menu para interação com o usuário e **Menu::parseCommand()** retorna as opções selecionadas pelo usuário seguindo a interface com o script Python. Além disso, a classe “**FileResolver.hpp**” implementa uma solução simples para ler o arquivo **log.tmp**.



```
Menu.hpp

class Menu
{
private:
    // Lida com selecoes booleanas (_faces, _3x4, _smooth, _detail)
    bool _handleBool(string msg);
    // Lida com selecao de floats (_color, _contrast)
    float _handleSelection(string msg);

    // Permite apenas entradas validas
    int _validateInput(int num, int min, int max);
    // Sobrecarga para o caso de float (_color, _contrast)
    float _validateInput(float num, float min, float max);

    // Inicializa a URI da classe
    void _handleURI();

    // Implementacao de std::cin para receber apenas valores int
    int _handleNumber();

public:
    // Instancia a classe e garante que todos os valores iniciem "zerados"
    Menu();
    // Inicializa o menu
    void run();
    // Retorna os parametros convertidos para a interface do script python
    string parseCommand();
};
```

Figura 4: Fragmento da implementação do cabeçalho da classe Menu.

b. Scripts Python

A segunda parte consiste na implementação de um script Python **interface.py** e uma classe **FileResolver.py**. O script **interface.py** é responsável por lidar com a usabilidade do lado do script, enquanto a classe **FileResolver** implementa os métodos para manipulação e download das imagens.

```

interface.py

# Importando a classe criada em ./ImageResolver.py
from ImageResolver import ImageResolver

# Biblioteca para uso das diretivas argc e argv
import sys

if __name__ == "__main__":
    uri, faces, _3x4, smooth, detail, color, contrast = sys.argv[1:]
    color, contrast = float(color), float(contrast)

    # Instancia classe e carrega a URI
    ir = ImageResolver(uri)
    # Baixa a imagem e armazena o BLOB na classe
    im = ir.loadImageFromWeb()

    if (faces == 'true'):
        # Mapeia os rostos na imagem
        ir.mapFaceRect()
        # Corta os rostos mapeados e cria o array im
        ir.cropImage()

        if (_3x4 == 'true'):
            im = ir.croppedIm3x4
        else:
            im = ir.croppedIm

    if (smooth == 'true'):
        im = map(ir.smoothFilter, im)

    if (detail == 'true'):
        im = map(ir.detailFilter, im)

    if (color != 1.0):
        # A funcao map espera que todos os argumentos da funcao de iteracao
        # sejam vetores de mesmo comprimento, mas como o valor do modificador e
        # uma constante utilizo uma lambda function para abstrair o segundo
        # parametro da funcao anonima passada como argumento da funcao map
        im = map(lambda arg: ir.colorFilter(im=arg, mod=color), im)

    if (contrast != 1.0):
        im = map(lambda arg: ir.contrastFilter(im=arg, mod=contrast), im)

    # Salvando os blobs como png
    imName = [ir.saveAs(im=name, index=i) for i, name in enumerate(im)]

    # Ecoando os nomes para o sistema, dessa forma poderao ser redirecionados
    # pelo operador '>' do UNIX
    sys.stdout.write('\n'.join(imName))

```

Figura 5: Fragmento da implementação da interface Python com a CLI em C++.

6. Conclusão

A implementação da CLI em C++ garante portabilidade e robustez, além de garantir a passagem correta dos parâmetros para o script Python de maneira segura. A implementação da interface via chamada de sistema é uma boa alternativa para esse caso, tendo em vista que o programa executa uma chamada por vez, o que não exige uma interface dinâmica entre as duas linguagens.

O uso de uma linguagem de alto nível como o Python possibilitou agilidade e flexibilidade na implementação do script tendo em vista a vasta quantidade de bibliotecas disponíveis para uso. Por fim, a integração entre linguagens de programação é uma ferramenta poderosa para construção de aplicações em C++ mais robustas e com menos código. Tendo em vista, que outras linguagens podem fornecer abstrações que simplificam algumas implementações que seriam mais complicadas e demoradas na linguagem.