**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**ĐHQG TP HỒ CHÍ MINH**

Logo

Description automatically generated

**BÁO CÁO ĐỒ ÁN GIỮA KÌ**

**BITMAP**

**Sinh viên thực hiện:** Đặng An Nguyên - 21120508

Contents

[Cấu trúc file header 3](#_Toc103635615)

[Cấu trúc các hàm trong file function.h 5](#_Toc103635616)

[Hàm đọc file BMP 5](#_Toc103635617)

[Hàm convert\_data 5](#_Toc103635618)

[Hàm ghi file BMP 5](#_Toc103635619)

[Hàm grayscale 6](#_Toc103635620)

[Hàm avg 7](#_Toc103635621)

[Hàm zoom 7](#_Toc103635622)

[Hàm readcolortable 8](#_Toc103635623)

[Cấu trúc file Source.cpp 8](#_Toc103635624)

[Debug 10](#_Toc103635625)

# Cấu trúc file header

* Xây dựng cấu trúc ảnh BMP như sơ đồ trình bày dưới đây:

Diagram

Description automatically generated

* Trong đó :
  + Cấu trúc thành phần header ( 14 bytes )
    - Signature (2 bytes) : Chữ kí file ( “BM” hoặc )
    - Size (4 bytes): Kích thước file ảnh BMP
    - Rerseved 1 – 2 (4 bytes): Phần dành riêng
    - DataOffSet (4 bytes): Địa chỉ phần bắt đầu lưu giá trị điểm ảnh
  + Cấu trúc phần DIB ( 40 bytes )
    - DIB\_size: Kích thước phần DIB >= 40 bytes
    - Width (4 bytes): Số pixel theo chiều rộng
    - Height (4 bytes): Số pixel theo chiều dài
    - Planes (2 bytes): Số lớp màu (=1)
    - Bpp ( bíts per pixel) (2 bytes): Số lượng bit trong 1 pixel
  + Phần dư temp: Vì có file BMP sẽ có phần DIB có kích thước lớn hơn 40 bytes nên cần 1 mảng 1 chiều để lưu phần dư này
  + Data: Là mảng 2 chiều để lưu các pixel theo sơ đồ dưới đây. Mỗi pixel tương ứng sẽ có 4 điểm hoặc 3 màu tương ứng A , R , G ,B. Và mảng data chỉ lưu dũ liệu điểm ảnh **( không bao gồm padding bytes )** để thuận tiện cho việc xử lí liên quan đến điểm ảnh

Table

Description automatically generated

# Cấu trúc các hàm trong file function.h

## Hàm đọc file BMP

**void readBMP(const char\* filepath, Picture& img, char\*& data, char\*& colortable)**

* Đọc file BMP từ đường dẫn filepath cho trước, lưu các dữ liệu phần header, dib, temp (nếu có) vào trong img ( được khai báo với cấu trúc Picuture)
* Mảng data 1 chiều để lưu các dữ liệu điểm ảnh và phần padding bytes ( nếu có )
* Colortable : Nếu ảnh là file BMP 8 bit, cấu trúc file 8 bit sẽ có phần khác với file 24 và 32 bit vì sẽ có colortable ( 1024 bytes ) sau phần DIB, và phía sau phần colortable sẽ là dữ liệu điểm ảnh của file BMP 8 bit.
* Tham số truyền vào:
  + Filepath: đường dẫn file BMP cần đọc
  + Img : Biến Picture để lưu dữ liệu về cấu trúc file BMP
  + Data: lưu dữ liệu điểm ảnh
  + Colortable : Nếu ảnh BMP là file 8 bit thì cần đọc thêm colortable

## Hàm convert\_data

**void convert\_data(Picture& img, char\* data)**

* Mục đích: Chuyển dữ liệu điểm ảnh trong mảng data đã lưu ở hàm readBMP thành mảng 2 chiều các pixel để thuận tiện trong quá trình xử lí.
* Tham số truyền vào:
  + Img: ảnh có data ( mảng 2 chiều ) để lưu dữ liệu điểm ảnh
  + Data: dữ liệu điểm ảnh từ hàm đọc readBMP

## Hàm ghi file BMP

* + - **Đối với file 24 bit và 32 bit**

**void writeBMP(Picture img, char\* data, const char\* output)**

* Tham số truyền vào:
  + Img: biến kiểu struct Picture lưu dữ liệu về header, dib, phần (temp) của ảnh BMP.
  + Data: Dữ liệu điểm ảnh dưới dạng mảng 1 chiều ( bao gồm cả padding bytes )
  + Output: Đường dẫn file ghi
    - **Đối với file 8 bit**

**void write8bitBMP(const char\* fileoutput, Picture img, char\* colortable, char\* data)**

* Tham số truyền vào:
  + Fileoutput: đường dẫn file ghi
  + Img: biến kiểu struct Picture lưu dữ liệu về header, dib, phần (temp) của ảnh BMP.
  + Colortable : Vì file 8 bit có phần colortable trước phần dữ liệu điểm ảnh
  + Data: Dữ liệu điểm ảnh dưới dạng mảng 1 chiều ( bao gồm cả padding bytes)

## Hàm grayscale

**Picture grayscale(Picture img, char\*& data8bit)**

* Sau khi thực hiện hàm sẽ trả về một ảnh bmp 8 bit
* Tham số truyền vào:
  + Ảnh cần chuyển thành 8 bit : img
  + Data8bit để lưu dữ liệu điểm ảnh
* Thuật toán: Giá trị trung bình của từng điểm ảnh trong file 8 bit được tính bằng trung bình cộng của 3 giá trị B G R trong file 24 hoặc 32 bit
* Mỗi pixel trong dữ liệu điểm ảnh chỉ lưu một giá trị duy nhất , khác với file 24 hay 32 bit lưu các giá trị A R G B.
* Sau khi tính toán xong giá trị điểm ảnh cho file BMP 8 bit. Ta tiến hành sao chép lại phần header và DIB của file gốc để tạo nên một cấu trúc file mới.
* Các thành phần chỉnh sửa lại trong file 8 bit:
  + BPP = 8;
  + Color used =256
  + Dataoffset = 1078 (14 bytes Header + 40 bytes DIB + 1024 bytes Colortable)

## Hàm avg

**Pix32 avg(Pix32\* a, int n)**

* Mục đích: Tính giá trị trung bình lần lượt của điểm ảnh A B G R và trả về 1 pixel lưu 4 giá trị màu trên.
* Tham số truyền vào :
  + Pix32\* a: Một mảng gồm n pixel
  + N: số lượng phần tử trong mảng pixel a

## Hàm zoom

**Picture zoom(const Picture src, int s, char\*& newdata)**

* Sau khi thực hiện hàm sẽ trả về một ảnh ( Picture ) đã được scale theo tỉ lệ s cho trước.
* Tham số truyền vào :
  + Src: ảnh cần dược zoom
  + S: tỉ lệ scale
  + Newdata: để lưu giá trị điểm ảnh mới ( bao gồm cả padding bytes)
* Thuật toán :
  + Sao chép lại phần header và dib của ảnh gốc và bắt đầu sửa lại các thông số sau:
    - Phần height và width dữ liệu điểm ảnh của ảnh mới = (height) / s+1
    - Table

      Description automatically generatedChia phần dữ liệu điểm ảnh của ảnh gốc thành các phần có kích thức s x s và tính trung bình các điểm ảnh A B G R bằng cách dùng hàm avg và gán giá trị đó vào 1 pixel trong ảnh mới
    - Lưu phần dữ liệu điểm ảnh ( có cả padding bytes ) vào trong mảng 1 chiều newdata ( để thuận tiện cho việc ghi dữ liệu )
    - Tính toán lại kích thước ảnh
  + Trả về ảnh zoom vừa được khởi tạo.

## Hàm readcolortable

* void readcolortable(char\*& colortable)
* Mục đích : đọc bảng màu từ file colortable.txt có trong thư mục debug và truyền vào mảng colortable được khởi tạo trong file Source.cpp

1. Hàm destructor

* Mục đích: Giải phóng vùng nhớ cấp phát động trong lúc thực hiện chương tình để tránh lỗi rò rỉ bộ nhớ ( memory leak )

# Cấu trúc file Source.cpp

* Sử dụng tham số dòng lệnh

int main(char argc, char\* argv[])

* Trong đó:
  + Argc: Số lượng tham số
  + Argv: mảng chứa các tham số đầu vào
    - Argv[0]: Đường dẫn chương trình
    - Argv[1]: Tên file bmp input
    - Argv[2]: tuỳ chọn -zoom hoặc -conv
    - Argv[3]: Tên file output
    - Argv[4]: Tỉ lệ scale nếu argv[2] là tuỳ chọn zoom
* Khởi tạo các biến
  + Img : lưu dữ liệu của ảnh bmp đầu vào
  + Dest: lưu dữ liệu ảnh bmp xuất ra
  + Data : lưu dữ liệu điểm ảnh ( bao gồm cả padding bytes) của file đầu vào
  + Newdata: lưu dữ liệu điểm ảnh ( bao gồm cả padding bytes) của file xuất ra
  + Colortable: được khởi tạo với 1024 phần tử kiểu char = 1024 bytes chứa bảng màu của file 8 bit được đọc bới hàm readcolortable từ 1 file có sẵn trong thư mục debug.
* Thực hiện lần lượt các việc sau:
  + Gọi hàm readcolortable để đọc bảng màu trong file colortable.txt
  + Gọi hàm readBMP để đọc file input đầu vào
  + Gọi hàm convert\_data để chuyển dữ liệu điểm ảnh thành mảng 2 chiều giúp cho việc xử lí trên các pixel dễ dàng hơn
  + Nếu strcpy(argv[2], “-zoom”)==0 thực hiện các việc sau:
    - Khởi tạo biến d để lưu tỉ lệ scale . Tỉ lệ scale được chuyển đổi từ chuỗi argv[4] thành số nguyên bằng hàm atoi trong thư viện string.h
    - Dùng biến dest để lưu dữ liệu trả về từ hàm zoom
    - Xuất file bmp từ dữ liệu biến dest bằng hàm writeBMP
  + Nếu strcpy(argv[2],”conv”)==0 thực hiện các việc sau:
    - Dùng biến dest để lưu dữ liệu ảnh trả về của hàm grayscale
    - Gọi hàm writeBMP để xuất ảnh vừa trả về
  + Gọi hàm destructor để giải phóng vùng nhớ
  + Giải phóng vùng nhớ cho colortable

# Debug

* Terminal: ./21120508.exe <a> <b> <c> <d>
* CMD: 21120508.exe <a> <b> <c> <d>
* Trong đó:
  + <a>: Là tên file BMP đầu vào. Ví dụ “lena.bmp”
  + <b>: Nếu chọn chế độ chuyển thành file 8 bit: “-conv”

Nếu chọn chế độ zoom :”-zoom”

* + <c>: Tên file BMP xuất
  + <d>: Tỉ lệ ảnh scale (nếu có)

**CHÚ Ý :** Thư mục chứa file exe phải có file colortable.txt