ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC TƯ NHIÊN

Khoa Công nghệ thông tin

-----🙞🙜🕮🙞🙜-----

**BÁO CÁO ĐỒ ÁN**

**Môn học: Thực hành Kỹ thuật lập trình**

**Đề tài: Xử lý ảnh Bitmap**

*Thành viên:*

Nguyễn Minh Triết  
Lâm Anh Toàn  
Trần Thanh Tịnh

**Mục lục**

[A. Giới thiệu Bitmap và một số thao tác cơ bản 5](#_Toc483296672)

[I. Giới thiệu 6](#_Toc483296673)

[II. Cấu trúc file Bitmap 6](#_Toc483296674)

[1. Bitmap header 7](#_Toc483296675)

[2. Bitmap information 7](#_Toc483296676)

[3. Color palette 8](#_Toc483296677)

[4. Bitmap data 9](#_Toc483296678)

[a) Bitmap data 9](#_Toc483296679)

[b) Padding 9](#_Toc483296680)

[III. Các thao tác cơ bản 10](#_Toc483296681)

[2. Chuyển dữ liệu đọc được thành mảng pixel 10](#_Toc483296682)

[3. Chuyển mảng pixel sang lại mảng data như ban đầu 11](#_Toc483296683)

[4. Ghi dữ liệu lên file Bitmap 12](#_Toc483296684)

[5. Cập nhật header 12](#_Toc483296685)

[B. Một số chức năng chỉnh sửa hình ảnh 13](#_Toc483296686)

[I. Cắt ảnh (crop image) 14](#_Toc483296687)

[II. Chỉnh sửa kích cỡ hình ảnh (resize image) 15](#_Toc483296688)

[1. Thu nhỏ ảnh 15](#_Toc483296689)

[2. Phóng to ảnh 16](#_Toc483296690)

[III. Chuyển thành ảnh trắng đen 17](#_Toc483296691)

[IV. Chuyển một màu trong ảnh thành màu khác 18](#_Toc483296692)

[1. Bằng màu của một pixel khác trong ảnh 18](#_Toc483296693)

[V. Điều chỉnh độ sáng và độ tương phản 19](#_Toc483296694)

[1. Độ sáng 19](#_Toc483296695)

[2. Độ tương phản 20](#_Toc483296696)

[VI. Xoay ảnh 21](#_Toc483296697)

[1. Xoay ảnh 90 độ 21](#_Toc483296698)

[2. Xoay ảnh 180 độ (lật ngược ảnh) 22](#_Toc483296699)

[3. Xoay ảnh 270 độ 22](#_Toc483296700)

[C. Tài liệu tham khảo 23](#_Toc483296701)

[D. Phụ lục 24](#_Toc483296702)

1. **Giới thiệu Bitmap và một số thao tác cơ bản**
2. **Giới thiệu**

Bitmap là một loại tập tin định dạng ảnh, thường có đuôi là .bmp và .dib.

Số bit trên một điểm ảnh (bits per pixel), ta sẽ tạm kí hiệu là **n**. Một ảnh Bitmap có 2n màu. Ví dụ:

* 1 bit: ảnh trắng đen
* 4 bit: ảnh 16 màu
* 8 bit: ảnh 256 màu
* 16 bit: ảnh 65536 màu
* 24 bit: ảnh khoảng 16 triệu màu

Điểm mạnh:

* Ảnh có thể được tạo dễ dàng bằng mảng dữ liệu pixel được lưu trữ sẵn.
* Có thể chuyển đổi giá trị các pixel dễ dàng bằng cách sử dụng tọa độ.
* Ảnh Bitmap có thể chuyển đổi tốt đối với các thiết bị đầu ra dạng điểm.

Điểm yếu:

* Kích thước file ảnh lớn.
* Việc thu nhỏ ảnh bằng cách bỏ qua các pixel và chỉ lấy các pixel ở tọa độ đã định trước sẽ không còn đảm bảo về chất lượng của bức ảnh.
* Có thể thu nhỏ ảnh bằng cách nén dữ liệu các pixel lại với nhau nhưng việc này rất tốn thời gian. Ảnh càng lớn thì quá trình càng lâu.

1. **Cấu trúc file Bitmap**

Do tập tin Bitmap có định dạng đơn giản nên bản thân chúng ta cũng có thể tự gõ được. Cấu trúc tập tin gồm có 4 phần:

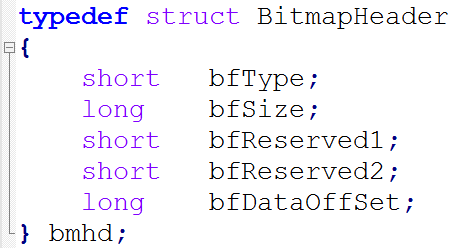
* Bitmap header (14 bytes): giúp nhận dạng file Bitmap.
* Bitmap information (40 bytes): chứa các thông tin chi tiết, thông số của ảnh giúp cho việc chỉnh sửa, xử lý ảnh.
* Color palette (4\*k bytes, với k là số màu của ảnh): định nghĩa các màu được sử dụng trong ảnh.
* Bitmap data: lưu dử liệu ảnh.

1. Bitmap header

Lưu ý: tên các biến các bạn có thể khai báo tùy ý sao cho phù hợp với mình nhất

Cấu trúc header như sau:

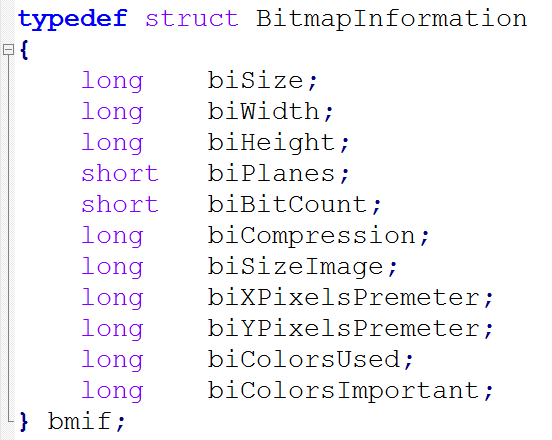
* **Type** (2 bytes): xác nhận đây có phải là file Bitmap hay không (chứa 2 kí tự là “BM” hoặc “0x4D42”).
* **Size** (4 bytes): kích thước của file (kích thước ảnh + kích thước header).
* **Reserved1** (2 bytes): không sử dụng.
* **Reserved2** (2 bytes): không sử dụng.
* **DataOffBit** (4 bytes): vị trí bắt đầu nội dung file, thường chứa giá trị là 54 (kích thước của Bitmap header và information, vì sau đó là đến nội dung dữ liệu của file)



1. Bitmap information

Cấu trúc header như sau:

* **Size** (4 bytes): kích thước phần còn lại của header ảnh.
* **Width** (4 bytes): chiều rộng của ảnh.
* **Height** (4 bytes): chiều cao của ảnh.
* **Planes** (2 bytes): số lượng plane màu sử dụng (phải bằng 1)
* **BitCount/BitPerPixel** (2 bytes): số lượng bit của 1 pixel, xác định độ phân giải của ảnh, là các giá trị 1, 4, 6, 16, 24 nói trên.
* **Compression** (4 bytes): loại nén
* **SizeImage** (4 bytes): kích thước của ảnh.
* **XPixelsPremeter** (4 bytes): độ phân giải theo chiều ngang.
* **YPixelsPremeter** (4 bytes): độ phân giải theo chiều dọc.
* **ColorUsed** (4 bytes): số lượng màu sắc sử dụng (bằng 0 tức là sử dụng tất cả các màu)
* **ColorImportant** (4 bytes): số lượng màu sắc quan trọng (cũng cho bằng 0)

****

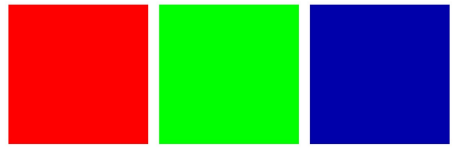
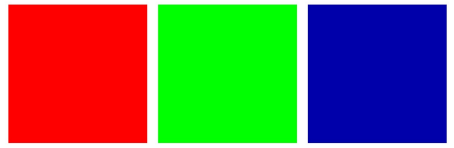
Về phần cấu trúc header, các bạn có thể viết tách ra làm hai phần như trên cho rõ rang hoặc viết gộp lại thành một cũng được, miễn làm sao đáp ứng 54 bytes cho header.

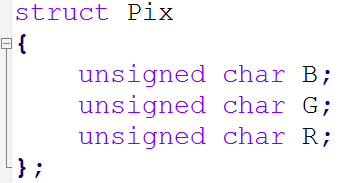
1. Color palette

Color palette gồm 4 byte sắp xếp liền nhau theo cấu trúc:

**Blue – Green – Red – Reverse**

Bảng màu của Bitmap lưu ngược lại so với bảng màu của màn hình nên khi đọc, ghi Bitmap cần chú ý thứ tự.



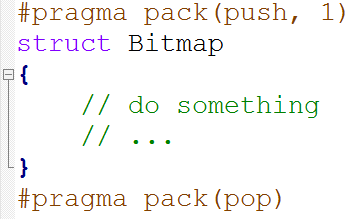


**CHÚ Ý:** Để tránh sự sai lệch về ảnh xảy ra, chúng ta phải thêm hai dòng dưới đây vào đầu và cuối cấu trúc header và cấu trúc pixel:





Ví dụ:



1. Bitmap data
2. *Bitmap data*

Data chứa các giá trị màu của các pixel trong ảnh. Các pixel được lưu từ trái sang phải và từ dưới lên trên. Mỗi byte trong vùng bitmap data biểu diễn 1 hoặc nhiều điểm ảnh tùy theo số bits trên 1 pixel.

Ví dụ data sẽ lưu theo dạng: 8 9 10 11

4 5 6 7

0 1 2 3

1. *Padding*

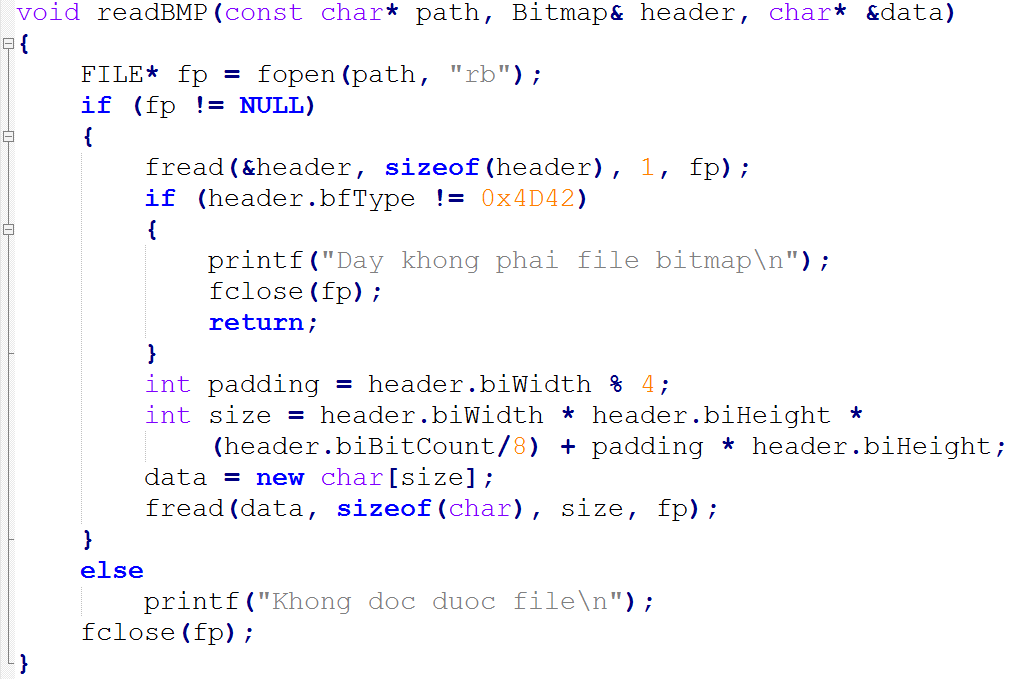
Do Bitmap yêu cầu chiều rộng là bội số của 4 mà khi nãy chúng ta chỉ sử dụng cấu trúc pixel có 3 bytes nên không phải lúc nào cũng đạt bội số của 4. Vì thế, ta cần phải có thêm các byte phụ để bù trừ chỗ trống còn thiếu đó, gọi là padding bytes. Padding chính là số dư của phép chia tổng tất cả các byte của chiều rộng với 4.

Ví dụ như một ảnh có chiều rộng là 20. Ta có 20 x 3 = 60 chia hết cho 4 nên padding = 0. Vòn một ảnh có chiều rộng là 10 thì 10 x 3 = 30 không chia hết cho 4 và dư 2 => padding = 2

1. **Các thao tác cơ bản**
2. Đọc file Bitmap

Để có thể đọc được tập tin Bitmap thì đầu tiên chúng ta phải mở file bằng nhị phân (read binary). Nếu như đây đúng là file Bitmap thì tiếp tục, nếu không, xuất thông báo lỗi, đóng file, thoát chương trình.

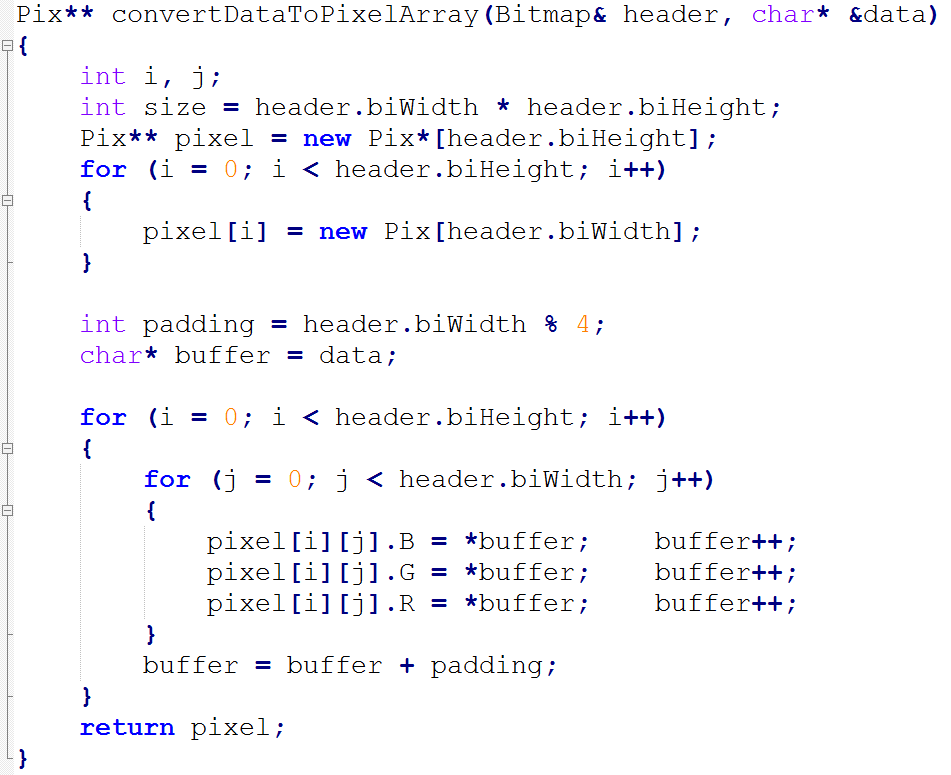
Nếu tiếp tục thì ta cần phải tính padding như đã nói trên, tính kích thước data. Kích thước data chính là tất cả các byte của ảnh cộng với cả tất cả các padding byte cần thiết. Ta sẽ tạo một mảng để chứa các dữ liệu đó dựa trên kích thước vừa tính được và đọc từ file Bitmap vào data.



Sau khi đọc xong, đừng quên đóng file bạn nhé.

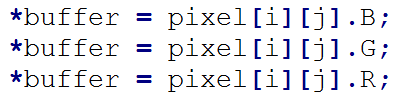
1. Chuyển dữ liệu đọc được thành mảng pixel

Thực ra thì thao tác này cũng giống như copy dữ liệu trong mảng này sang mảng khác thôi. Và chuyển dữ liệu đó qua định dạng theo cấu trúc pixel sẽ giúp ta thao tác với ảnh dễ dàng hơn. Các bạn có thể sử dụng mảng một chiều hoặc hai chiều tùy ý. Đối với các bạn không tốt lắm trong việc tính toán các chỉ số, tọa độ thì mình khuyên các bạn nên sử dụng mảng hai chiều.

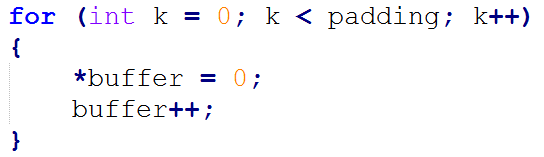


1. Chuyển mảng pixel sang lại mảng data như ban đầu

Thao tác này chẳng qua cũng y như thao tác trên và ngược lại thôi, chỉ cần thay thế vài dòng thôi.



Nhưng lưu ý thêm, trong thao tác trên, ta dùng: buffer = buffer + padding; để nhảy qua các byte bù đắp và tiếp tục sao chép. Còn trong thao tác này, chúng ta sẽ phải gán cho các vị trí bù đắp một giá trị nào đó, thường là 0 (cho thuận tiện).



1. Ghi dữ liệu lên file Bitmap

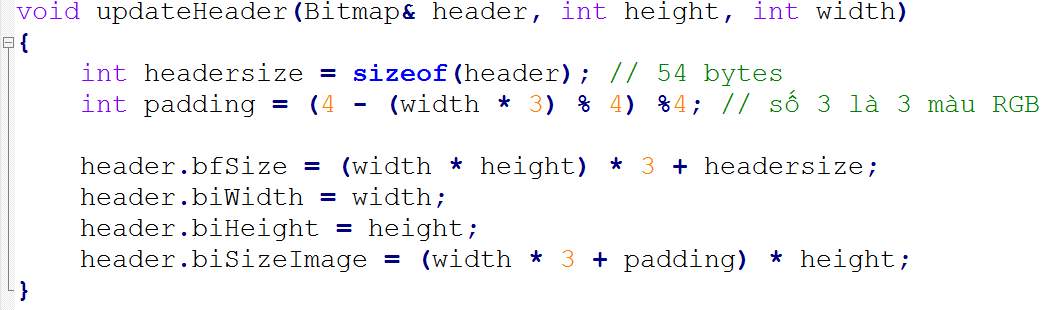
Cũng tương tự như đọc tập tin Bitmap với cách mở file là “wb” (write binary) và thay hàm fread thành fwrite. Lần này, vì chúng ta đang ghi dữ liệu thành file Bitmap nên tất nhiên là không cần kiểm tra đây có phải là file Bitmap không rồi.

1. Cập nhật header

Một trong những điều vô cùng quan trọng khi thao tác với Bitmap chính là cập nhật lại các thông tin của header để chúng trùng khớp với những gì bạn đã thay đổi. Nếu không cập nhật lại, thì ảnh sẽ bị biến dạng một kiểu nào đó. Đây giống như việc “Lấy râu ông này cắm cằm bà kia” ấy.

Thường thì chúng ta chỉ thay đổi về chiều cao, chiều rộng của file, khi đó thì kích thước file cũng sẽ thay đổi nên cũng cần phải tính toán lại. Bạn có thể viết một hàm như updateHeader với các tham số truyền vào là những điều bạn đã thay đổi, về phần tính toán thì bạn có thể tính lại trong hàm, hoặc truyền vào sau khi đã tính được trong phần thay đổi.

Ví dụ:

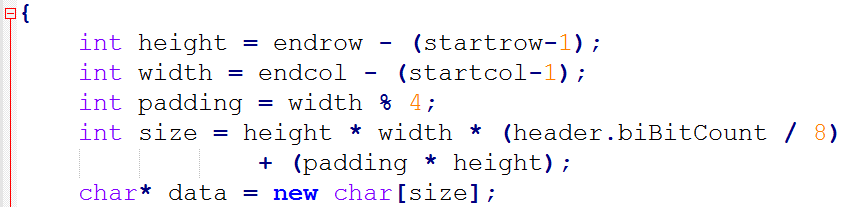


1. **Một số chức năng chỉnh sửa hình ảnh**
2. **Cắt ảnh (crop image)**

Để cắt hình ảnh hết sức đơn giản, bạn chỉ việc đọc ảnh, chuyển đúng phần dữ liệu mình cần (nhập từ bên ngoài vào) trong mảng pixel vào mảng data rồi ghi thành file bitmap là xong. Nó ý như việc bạn lấy một phần thông tin nào đó trong một ma trận bình thường thôi, không có gì đặc biệt cả. Nhớ cẩn thận trong việc tính toán các vị trí bắt đầu, kết thúc cho chính xác, và đừng quên update header sau khi cắt nhé.







Tới đây bạn cho duyệt ma trận pixel với 2 biến chạy từ startrow đến endrow và startcol đến endcol rồi đưa lần lượt vào mảng data nhé, chú ý thứ tự BGR. Nhớ cập nhật lại header và return mảng data ra ngoài.



Ví dụ: Height: 100 -> 200, width: 150 -> 300



1. **Chỉnh sửa kích cỡ hình ảnh (resize image)**

Trước tiên mình muốn nói rằng hiện tại, mình chỉ làm được thay đổi số lần là số nguyên thôi, như 200%, 300%, … chứ chưa làm được các số liệu như 125%, 180%, …

1. Thu nhỏ ảnh

Và trong thu nhỏ ảnh, mình vẫn chưa làm được cách nén các pixel lại với nhau thành 1 pixel nên mình chỉ làm cách chỉ lấy các pixel ở các vị trí cách nhau số ô là số lần nhập vào. Cách này là không hay, vì nó sẽ bỏ qua rất nhiều pixel dẫn đến hình sẽ không còn rõ như trước, méo mó, …

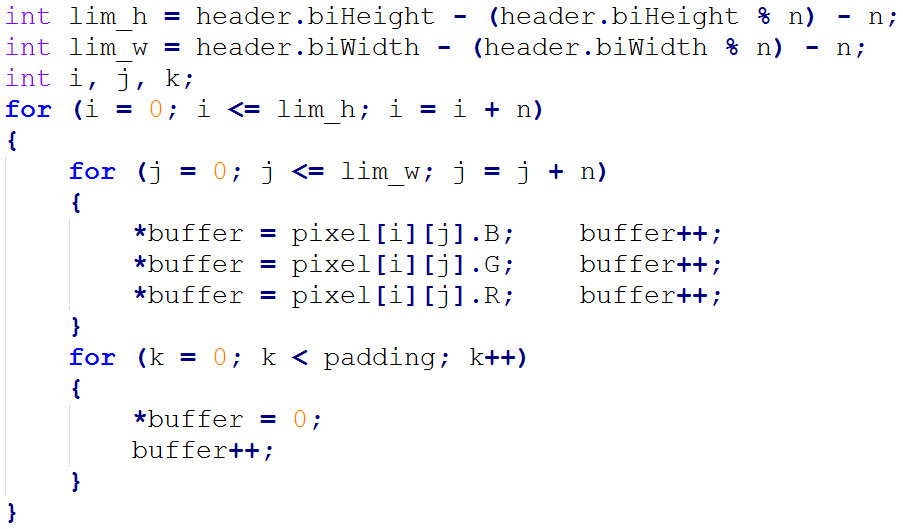
Ví dụ như dữ liệu trong dòng đó là 0, 1, 0, 3, 0, 0, 2, 0 và chúng ta muốn giảm xuống 2 lần, tức là: (0, 0, 0, 2). Như các bạn đã thấy, các dữ liệu quan trọng như 1, 3 đã không lấy được bằng cách này. Khi thu nhỏ đến một mức độ nào đó, một vài chi tiết trong hình có thể sẽ biến mất.

Nay mình chỉ hướng dẫn với mục đích tham khảo chứ không khuyến cáo các bạn sử dụng phương pháp này.

Có nhiều cách tính toán khác nhau, thì mình sẽ làm mẫu một cách cho các bạn. Chúng ta vẫn cho vòng lặp chạy duyệt cả ma trận với i, j lần lượt là biến chỉ số dòng, cột. Lưu ý rằng sẽ có khi kích thước ảnh không chia hết cho số lần thu nhỏ nhập vào, và chúng ta sẽ bỏ luôn các dòng dư đó. Bằng cách cho chiều rộng trừ số dư của chiều rộng và số lần thu nhỏ trừ số lần thu nhỏ <width – (width mod n) – n>, tương tự cho chiều cao. Và sau khi duyệt một ô thì j = j + n, sau một dòng thì i = i + n.

\*Giải thích: Giả sử chiều rộng là 421, số lần thu nhỏ là 2. Khi chia 2 thì số nguyên thu được là 240, tức là trong một dòng, ta phải duyệt được 240 pixel. Nếu để j chạy từ 0 -> width, tức là 0 -> 421, j sẽ chuyệt qua 241 ô (dư 1 ô). Còn nếu áp dụng công thức: 421 - 1 - 2 = 418, với 0 <= j <= 418, j = j + 2 thì j sẽ duyệt đúng 240 ô.

Code:



Hiển nhiên là chúng ta phải cập nhật lại header sau khi hoàn tất chỉnh sửa, về sau các bạn phải luôn nhớ điều này, mình sẽ không nhắc nữa.

1. Phóng to ảnh

Làm ngược lại với thu nhỏ ảnh. Thay vì gộp nhiều pixel lại thành 1 thì bây giờ chúng ta ghi 1 pixel nhiều lần.

Ví dụ: Chúng ta có ma trận sau và số lần phóng to là 2

2 3 5 1 2 2 3 3 5 5 1 1

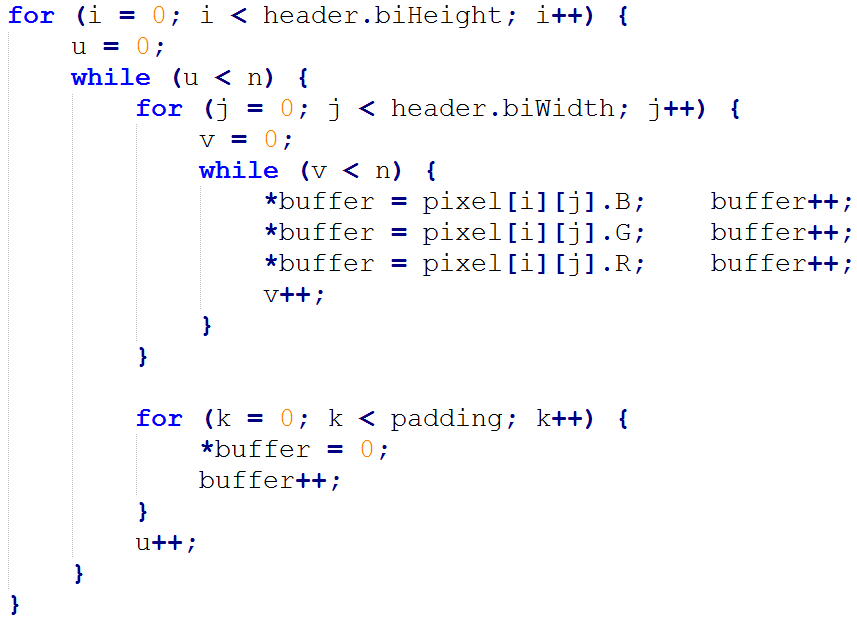
4 9 8 7 2 2 3 3 5 5 1 1

4 4 9 9 8 8 7 7

4 4 9 9 8 8 7 7

Chắc các bạn cũng hình dung ra thuật toán rồi, trong một dòng, ghi n lần với mỗi pixel đọc được, và ghi n lần dòng đó, sau đó mới đọc sang dòng tiếp theo.

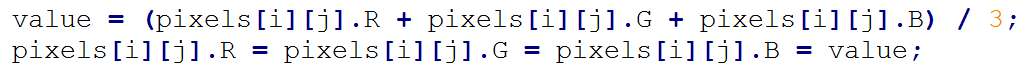
Code:



1. **Chuyển thành ảnh trắng đen**

Thao tác này hết sức đơn giản, tất cả những gì bạn cần làm là gán giá trị mỗi loại màu bằng trung bình cộng giá trị ba màu lại với nhau. Nó cũng tương tự như ngoài đời, khi bạn trộn ba màu chủ đạo lại với nhau thì chúng sẽ trở thành màu đen, và tùy theo độ sáng tối của mỗi màu mà ta có được màu xám đậm, xám nhạt, …

Cho i, j chạy duyệt toàn ma trận pixel và thay giá trị của chúng như sau:





1. **Chuyển một màu trong ảnh thành màu khác**

Nếu dùng các hình quá phức tạp, hỗn hợp rất nhiều màu sắc khác nhau thì khó có thể thấy được sự khác biệt. Vì chỉ cần một sai lệch nhỏ thôi thì đã được xem là màu khác rồi. Ví dụ: RGB (223, 190, 61) khác RGB (222, 190, 61).

1. Bằng màu của một pixel khác trong ảnh

Trước tiên, chúng ta cần phải xác định màu cần được đổi bằng cách nhập tọa độ pixel và lưu vào một biến (tạm gọi là “src”) có cấu trúc 3 màu (struct Pix ở trên). Sau đó bằng cách tương tự, ta xác định màu để đổi (tạm gọi là “dest”).

Tiếp theo, duyệt qua cả ma trận pixel, nếu như 3 màu của 1 ô mà trùng với 3 màu của src thì khi ghi vào data sẽ là dest. Code rất đơn giản, chỉ việc thay thế các pixel thôi, chắc chắn là các bạn làm được.

[Hình ảnh minh họa](#Phụ_lục_24) (phần phụ lục trang 24)

1. Bằng màu trong bảng màu

Với phương pháp này thì hiển nhiên chúng ta có được nhiều lựa chọn màu sắc hơn. Các bạn có thể sử dụng bảng màu có sẵn trong window. Tham khảo bảng màu tại [đây](https://minhthomson.wordpress.com/2011/04/29/b%E1%BA%A3ng-ma-mau-rgb-full/). Lưu ý là bảng màu trong window thì theo thứ tự là RGB. Còn tất cả thao tác thì đều giống như cách trên.

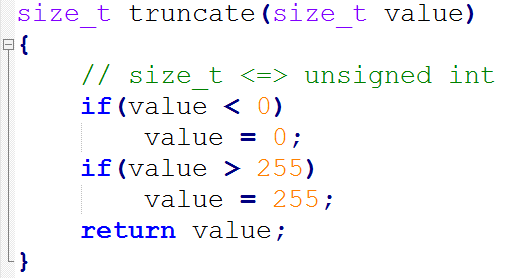
[Hình ảnh minh họa](#Phụ_lục_24) (phần phụ lục trang 24)

1. **Điều chỉnh độ sáng và độ tương phản**

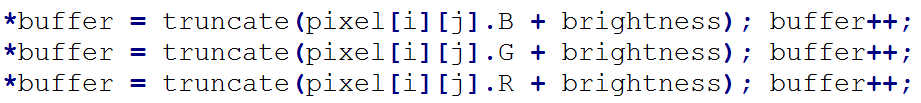
Về độ sáng và độ tương phản, chúng ta sẽ cần thêm một hàm hỗ trợ để giá trị màu sắc không rơi vào các vùng < 0 hoặc > 255 (sẽ nói rõ hơn trong từng phần)

1. Độ sáng

Với việc thay đổi độ sáng, chúng ta cần một hàm như nói trên với kiểu dữ liệu là số nguyên.



Và chúng ta chỉ đơn giản là cộng giá trị hiện có của lần lượt các loại màu với độ sáng đã nhập vào trước đó rồi cho hàm truncate duyệt qua, sau đó gán cho màu tương ứng.



Thường thì giá trị độ sáng nhập vào nằm từ -255 đến 255.

Ví dụ: brightness = 100



1. Độ tương phản

Cũng như điều chỉnh độ sáng, chúng ta cần một hàm hỗ trợ để giá trị màu luôn nằm trong phạm vi từ 0 đến 255. Nhưng độ tương phản có đôi chút khác biệt, kiểu trả về của hàm là số thực chứ không phải số nguyên. Vì để thấy được nhiều sự thay đổi hơn thì số thực sẽ biểu diễn chính xác hơn. Trong khi nếu dùng số nguyên thì chúng ta sẽ chỉ có được một số ít sự thay đổi thôi.

Còn về việc tính toán, tất cả đã có công thức, việc của chúng ta chỉ là áp dụng. Các kí hiệu: - F (factor): hệ số (kiểu số thực)

- C (contrast): mức tương phản nhập vào (số nguyên)

- B (blue): giá trị màu xanh của 1 ô pixel

- new\_B: giá trị màu xanh khi thay đổi độ tương phản của 1 ô pixel

Thường thì -255 <= C <= 255 như độ sáng. Nhưng giá trị âm sẽ làm cho hình thành màu xám

Ta có: F = [259 x (255 + C)] / [255 x (259 – C)]

Sau đó: new\_B = F x (B – 128) + 128, tương tự cho 2 màu còn lại

Rồi chúng ta lọc giá trị qua truncate và đưa vào data.

Ví dụ: C1 = 100, C2 = -10



C1



C2

1. **Xoay ảnh**

Tất cả các thao tác xoay ảnh đều tuân theo chiều kim đồng hồ. [Hình minh họa](#Phụ_lục_25)

1. Xoay ảnh 90 độ

Như đã đề cập ở phần A, các pixel được lưu theo từ trái sang phải, từ dưới lên trên.

Ta có: 8 9 10 11 Ta cần: 0 4 8

4 5 6 7 1 5 9

0 1 2 3  2 6 10

3 7 11

Các bạn có thể cho xoay ma trận pixel trước như ma trận bình thường rồi sau đó chép vào mảng data theo chiều thuận từ trái sang phải, từ trên xuống dưới.

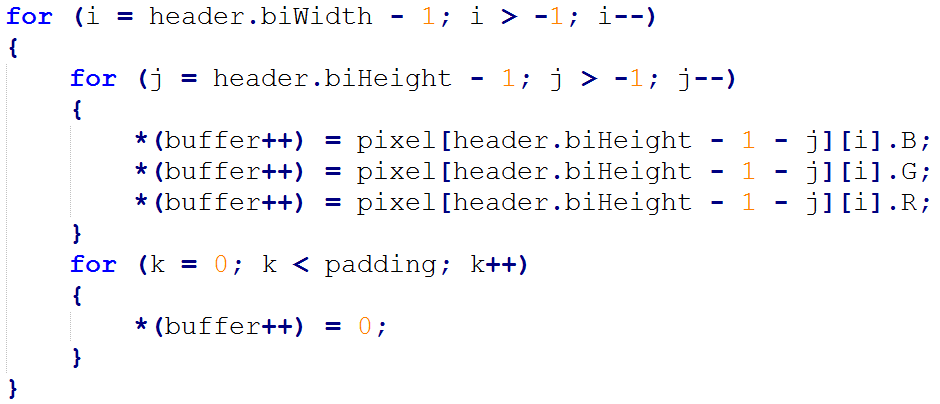
Hoặc là các bạn có thể ghi trực tiếp vào mảng data bằng cách lần lượt lấy các phần tử theo từng cột từ dưới lên, rồi sang cột tiếp theo.

8 9 10 … 0 4 8 1 5 9 2 …

4 5 6

0 1 2

Lần này mình sẽ cho các bạn xem code, trong các loại xoay ảnh tiếp theo, các bạn hãy tự tính toán cho mình cách lưu các pixel vào data. Việc đó không khó thế đâu.



1. Xoay ảnh 180 độ (lật ngược ảnh)

Để lật ngược ảnh thì chúng ta đảo ngược toàn ma trận, tức là đảo về cả dòng và cột.

Ta có: 8 9 10 11 Ta cần: 3 2 1 0

4 5 6 7 7 6 5 4

0 1 2 3 11 10 9 8

Tức là lấy lần lượt các phần tử ở dòng cuối cùng từ cột cuối trở ra trước. Rồi chuyển lên các dòng trên và tương tự.

8 9 10 11

4 5 6 7 3 2 1 0 7 6 5 …

0 1 2 3

1. Xoay ảnh 270 độ

Thực chất xoay ảnh 270 độ chẳng qua là xoay ảnh 90 độ ngược chiều kim đồng hồ, tức là chỉ cần đảo các cột của xoay 90 độ cùng chiều kim đồng hồ là xong.

Ta có: 8 9 10 11 Ta cần: 8 4 0

4 5 6 7 9 5 1

0 1 2 3 10 6 2

11 7 3

Hoặc ghi trực tiếp bằng cách ghi ngược lại của cách ghi 90 độ theo chiều kim đồn hồ nói trên. Tức là lấy theo cột lần lượt từ trên xuống rồi chuyển sang các cột tiếp theo và tương tự.

8 9 10 … 8 4 0 9 5 1 10 …

4 5 6

0 1 2

1. **Tài liệu tham khảo**

* Bitmap format

<https://en.wikipedia.org/wiki/BMP_file_format>

* Các khái niệm và xử lý cơ bản

<https://tinyurl.com/kupb9fh>

* TÌm hiểu thêm về các kiểu tạo nên hình ảnh của Bitmap

<https://tinyurl.com/meuwdnp>

* Một số thao tác về chỉnh sửa ảnh

<https://tinyurl.com/lhbxlcd>

* Độ sắc bén (rõ nét) của hình ảnh

<https://en.wikipedia.org/wiki/Acutance>

* Bài tổng hợp về thu nhỏ hình ảnh

<https://tinyurl.com/mjggwwm>

* The rule of three

<https://tinyurl.com/a7dstmx>

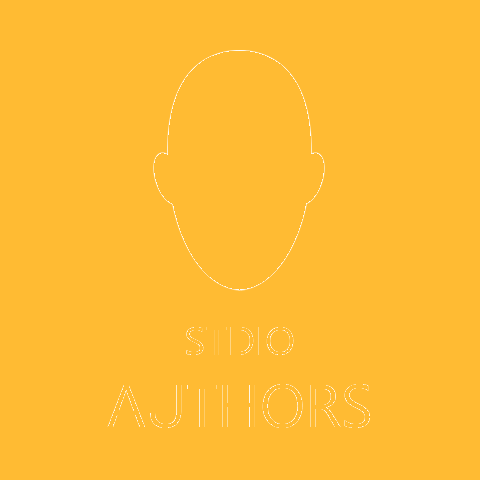
* Sử dụng C#

<https://tinyurl.com/m2n9ovg>

1. **Phụ lục**

* Phần đổi màu hình ảnh

Cách 1: Hình ảnh minh họa cho cách lấy màu của pixel khác trong hình: src(250,70), dest(1,1). [Trở lại](#Hình_đổi_màu_1)



Cách 2: Hình ảnh minh họa cho việc sử dụng bảng màu: src(1, 1), dest: màu xanh lá (đúng hơn là Lime). [Trở lại](#Hình_đổi_màu_1)



* Phần xoay ảnh



90 độ

270 độ

180 độ

[Trở lại](#Hình_xoay)