BÁO CÁO ĐỒ ÁN

BÀI TẬP LÝ THUYẾT 1

THAO TÁC TẬP TIN BMP

Người thực hiện:
Nguyễn Phước Anh Tuấn – 21120588
21120588@student.hcmus.edu.vn

I. Tổng quan:

Trong bài tập lý thuyết 1, dưới sự hướng dẫn của thầy Phạm Minh Hoàng, em đã thực hiện các thao tác cơ bản trên tập file BMP (Bitmap Image File).
Chương trình bao gồm các tính năng: Đọc và ghi file BMP, chuyển file BMP dạng 24/32 bit sang dạng file BMP 8 bit (ảnh đa xám) và thu nhỏ file BMP dạng 8/24/32 bit.
Chương trình bao gồm 1 file – source.cpp, chứa tất cả các hàm nhập xuất và thao tác trên file.

Hình 1. Các struct dùng trong bài

```
//Cấu trúc màu BGR/ABGR cho ảnh 24/32 bit
struct colorPix {
    unsigned char A;
    unsigned char B;
    unsigned char G;
    unsigned char R;
};
//Cấu trúc bảng màu cho ảnh 8bit
struct colorTable {
    uint8_t B;
    uint8_t G;
    uint8_t R;
    uint8_t Reserved;
};
```

Hình 2. Struct colorPix và colorTable

Trong chương trình này, em đã thêm 2 struct đặc biệt. Một là **struct colorPix{}**, dùng để chứa giá trị màu của 1 pixel trong ảnh 24/32bit (dùng để làm câu 4). Hai là **struct colorTable{},** phục vụ cho thao tác ảnh 8 bit, chi tiết công dụng của **colorTable{}** sẽ được nêu chi tiết ở các phần sau.

II. Các hàm cụ thể:

```
48  //Hàm đọc file bmp
49  int readBMP(const char* filename, BMPImage& bmp) {
50      ifstream fin(filename, ios::binary | ios::in);
51      if (!fin)
52          return 0;
53      //Đưa con trỏ về vị trí đầu file
54      fin.seekg(0, fin.beg);
55      //Đọc header (14 bytes)
56      fin.read((char*)&bmp.header, 14);
57      //Đọc DIB (40 bytes)
58      fin.read((char*)&bmp.dib, 40);
59      //Kiểm tra phần thừa
60      if (bmp.dib.dibSize > 40) {
61          //Kích thước phần dư
62          int remainder = bmp.dib.dibSize - 40; //Trừ đi phần đã đọc
63          //Nếu dư thì cấp phát bộ nhớ
64          bmp.pDIBReserved = new char[remainder];
65          //Đọc dữ liệu từ file cho phần dư
66          fin.read(bmp.pDIBReserved, remainder);
67      }
68      else
69          fin.read(bmp.pDIBReserved, 0);
70      //Đọc bảng màu nếu bpp bé hơn bằng 8bpp
71      if (bmp.dib.bpp <= 8) {
72          fin.read((char*)bmp.colorTable, sizeof(bmp.colorTable));
73      }
74      //Cấp phát vùng nhớ cho phần pixel data
75      bmp.pImageData = new char[bmp.dib.imageSize];
76      //Đọc phần dữ liệu điểm ảnh
77      fin.read((char*)bmp.pImageData, bmp.dib.imageSize);
78
79      fin.close();
80      return 1;
81
82  }
```

Hình 3. Hàm đọc file BMP

Ở hàm **readBMP()**, ta đầu tiên kiểm tra xem file đã được mở chưa. Nếu mở thành công, ta sẽ cho con trỏ file về vị trí đầu file. Sau đó ta đọc lần lượt các thông tin như **bmp.Header** (14 bytes), **bmp.dib** (40 bytes), ở đây nếu như phần bmp.dib.dibSize bị dư ra (tức là lớn hơn 40) thì ta sẽ lưu phần dư đó vào **bmp.pDIBReserved**. Chính vì ở câu 4 yêu cầu thu nhỏ ảnh 8 bit, nên ta cần xét trường hợp nếu như ảnh **SOURCE** là 8 bit. Từ đó ta sẽ đọc bảng màu của nó với kích thước bằng **256 * 4** (4 byte cho B G R và Reserved). Cuối cùng ta đọc dữ liệu điểm ảnh **bmp.pImageData** và đóng file

```
//Hàm ghi file bmp
int writeBMP(const char* filename, BMPImage bmp) {
    ofstream fout(filename, ios::binary | ios::out);
    if (!fout)
        return 0;
    //Đưa con trỏ về vị trí đầu file
    fout.seekp(0, fout.beg);
    //Ghi header
    fout.write((char*)&bmp.header, 14);
    //Ghi dib
    fout.write((char*)&bmp.dib, 40);
    //Ghi phần dư dib (nếu có)
    if (bmp.dib.dibSize > 40)
        fout.write(bmp.pDIBReserved, bmp.dib.dibSize - 40);
    //Nếu bpp <= 8 thì ghi phần bảng màu vào
    if (bmp.dib.bpp <= 8) {
        fout.write((char*)bmp.colorTable, sizeof(bmp.colorTable));
    }
    //Ghi phần dữ liệu điểm ảnh
    if (bmp.pImageData == NULL)
        return 0;

    fout.write((char*)bmp.pImageData, bmp.dib.imageSize);
    //Đóng file
    fout.close();

    return 1;

}
```

Hình 4. Hàm ghi file BMP

Tương tự như ở hàm đọc file, ở hàm **writeBMP()**, ta ghi tuần tự các thông tin như **bmp.Header** (14 bytes), **bmp.dib** (40 bytes) và **phần dư** (nếu có). Sau đó, vì ta cần phải ghi ảnh 8 bit ở cả câu 3 và 4 nên ta cần xét trường hợp bpp của ảnh dưới 8 bit. Từ đó có thể ghi bảng màu (**bmp.colorTable**) tương ứng với kích thước bằng **256 * 4**. Cuối cùng ta ghi dữ liệu điểm ảnh **bmp.pImageData.**

```
//Hàm chuyển đổi sang ảnh 8 bit
int to8Bit(BMPImage srcBmp, BMPImage& desBmp) {
    if ((srcBmp.dib.bpp != 24 && srcBmp.dib.bpp != 32) || srcBmp.pImageData == NULL)
        return 0;
    //Gán các thông tin quan trọng
    desBmp.header = srcBmp.header;
    desBmp.dib = srcBmp.dib;
    //cho desBpp = 8
    desBmp.dib.bpp = 8;
    //Gán pDIBReserved nếu có
    if (desBmp.dib.dibSize > 40) {
        int remainder = desBmp.dib.dibSize - 40;
        desBmp.pDIBReserved = new char[remainder];
    }
    //Cho kích thước = trị tuyệt đối (để tính toán)
    int height = abs(srcBmp.dib.height);
    int width = abs(srcBmp.dib.width);

    //Tính tổng số byte của 1 dòng width
    int lineBytes = 0;
    //Nếu bpp = 24 bits/pixel thì nhân 3
    if (srcBmp.dib.bpp == 24)
        lineBytes = width * 3;
    //Nếu bpp = 32 bits/pixel thì nhân 4
    if (srcBmp.dib.bpp == 32)
        lineBytes = width * 4;

    int newLineBytes = lineBytes;
    //Kiểm tra tổng số byte trên 1 dòng có là bội số của 4 không
    if (lineBytes % 4 != 0) {
        //Nếu chia 4 dư khác 0 thì
        //Cho số chia + số bị chia - số dư để được số byte là bội của 4
        newLineBytes = lineBytes + 4 - (lineBytes % 4);
    }

    //Tính padding byte (32bpp không có padding bytes)
    int padding = newLineBytes - lineBytes;
```

Hình 5.1. Hàm chuyển sang 8 bit

```
    //Tính kích thước phần pixel data
    desBmp.dib.imageSize = (width * height * desBmp.dib.bpp) / 8 + padding * height;
    //Cấp phát vùng nhớ cho pixel data của ảnh desBmp
    desBmp.pImageData = new char[desBmp.dib.imageSize];
    //Con trỏ pRow trỏ đến pImageData
    char* pSrcRow = srcBmp.pImageData;
    char* pDesRow = desBmp.pImageData;
    //Số byte của 1 pixel ảnh
    int nSrcByteInPix = srcBmp.dib.bpp / 8;
    int nDesByteInPix = desBmp.dib.bpp / 8;
    //Số byte trên 1 dòng của ảnh
    int nSrcByteInRow = (width * nSrcByteInPix) + padding;
    int nDesByteInRow = (width * nDesByteInPix) + padding;
    //Con trỏ từng pixel
    char* pSrcPix;
    char* pDesPix;
    //Tạo bảng màu cho ảnh 8 bit
    //vì khi chuyển từ bpp cao hơn sang 8 bit
    //Dữ liệu màu bị mất đi, ta cần tạo bảng màu mới cho ảnh
    for (int i = 0; i < 256; i++) {
        desBmp.colorTable[i].B = i;
        desBmp.colorTable[i].G = i;
        desBmp.colorTable[i].R = i;
        desBmp.colorTable[i].Reserved = 0;
    }
    //Vòng lặp đi qua từng dòng và cột
    for (int y = 0; y < height; y++) {
        //Khởi tạo biến màu
        //Vì là 8 bit nên ta dùng uint8_t
        uint8_t B, G, R;
        uint8_t aveColor;
        //Tại dòng thứ y: đi qua từng pixel trên dòng
        //Con trỏ pDesPix nhảy vào từng pixel
        pSrcPix = pSrcRow;
        pDesPix = pDesRow;
        //Truy xuất từng byte của mỗi pixel
        for (int x = 0; x < width; x++) {
            if (srcBmp.dib.bpp == 24) {
                B = pSrcPix[0];
                G = pSrcPix[1];
                R = pSrcPix[2];
            }
```

Hình 5.2. Hàm chuyển sang 8 bit

```
            if (srcBmp.dib.bpp == 32) {
                B = pSrcPix[1];
                G = pSrcPix[2];
                R = pSrcPix[3];
            }
            //Tính trung bình cộng 3 màu R G B
            aveColor = (B + G + R) / 3;
            //Gán giá trị màu cho từng pixel pDesPix đi vào
            pDesPix[0] = aveColor;
            //p...Pix nhảy qua pixel kế tiếp
            //= công thêm 1 khoảng số byte trong 1 pixel
            pSrcPix += nSrcByteInPix;
            pDesPix += nDesByteInPix;
        }
        //Con trỏ nhảy qua dòng tiếp theo = cộng thêm 1 lượng
        //byte trên 1 dòng (kể cả padding)
        pSrcRow += nSrcByteInRow;
        pDesRow += nDesByteInRow;
    }
    return 1;
}
```

Hình 5.3. Hàm chuyển sang 8 bit

Trong hàm **to8Bit()** sẽ truyền tham chiếu cấu trúc **desBmp**. Đầu tiên ta cần kiểm tra định dạng file đã đúng chưa. Sau đó ta sẽ gán các thông tin quan trọng trước (**header** và **dib**). Vì ta đang chuyển sang ảnh 8 bit nên ta sẽ gán **desBmp.dib.bpp = 8**. Ta cũng sẽ lưu phần dữ (nếu có). Ở đây, ta tạo 2 biến mới là **height** và **width** lần lượt là các giá trị tuyệt đối của kích thước ảnh **SOURCE** (để thuận tiện trong việc tính toán). Sau đó ta sẽ tính **padding** bytes. Bài toán đặt ra vấn đề là mình phải xét từng pixel trong ảnh, để tránh việc sử dụng mảng 2 chiều, em đã khai báo lần lượt các con trỏ để trỏ đến các điểm pixel tương ứng. Trên 1 dòng, con trỏ sẽ liên tục nhảy qua một lượng byte trong 1 pixel **(nSrcByteInPix/nDesByteInPix)** và khi hết dòng thì con trỏ chỉ dòng **(pSrcRow/pDesRow)** sẽ nhảy một lượng byte trong 1 dòng (kể cả padding byte) **(nSrcByteInRow/nDesByteInRow).**

Khi ta chuyển ảnh 32/24 bit sang 8 bit, dữ liệu màu sẽ bị mất đi nên ta cần tạo một bảng màu mới cho nó. Bảng màu này sẽ bao gồm các màu mà ảnh sẽ sử dụng theo cấu trúc được khai báo trong struct **colorTable{}** (phần **Reserved** trong **colorTable** thường bằng 0). Giá trị màu của ảnh 8 bit sẽ được tính bằng trung bình cộng của ba màu, theo thứ tự , **B (blue), G (green)** và **R (red)**. Đối với ảnh 32 bit ta sẽ không xét giá trị **A (Alpha)** tương trưng cho độ transparent của màu. **pDesPix** là con trỏ nhảy vào từng pixel của ảnh Destination.

```
//Hàm tìm màu trung bình
unsigned char findAveColor(DIB dib, colorPix& p, char* pSrcRow, int nSrcByteInRow, int nSrcByteInPix, int S) {
    unsigned int aveCol = 0;
    unsigned int BGR;
    //Xét giá trị
    switch (dib.bpp) {
    //Trường hợp bpp == 8
    case 8: {
        aveCol = 0;
        for (int i = 0; i < S; i++) {

            char* pPix = pSrcRow;

            for (int j = 0; j < S; j++) {
                //Tính tổng các giá trị của pixel trong ô vuông
                aveCol += (unsigned char)pPix[0];
                //Di chuyển pPix sang pixel tiếp theo
                pPix += nSrcByteInPix;
            }
            //Di chuyển pSrcRow sang dòng tiếp theo
            pSrcRow += nSrcByteInRow;
        }
        //Gán lại giá trị aveCol = trung bình cộng của tổng các màu
        //Ép kiểu lại thành unsigned char (từ unsigned int)
        aveCol = (unsigned char)(aveCol / (S * S));
        //Trả lại giá trị aveCol (unsigned char)
        return aveCol;
        break;
    }
    //Trường hợp bpp == 24
    case 24: {
        //Khởi tạo mảng BGR và gán giá trị 0 để tránh bị tràn
        //dữ liệu khi tính toán
        unsigned int BGR[3]= {0};
        p.B = p.G = p.R = 0;
        for (int i = 0; i < S; i++) {
            //pPix trỏ đến từng dòng đang xét
            char* pPix = pSrcRow;
```

Hình 6.1. Hàm tìm màu trung bình (theo từng ô vuông size S)

```
            for (int j = 0; j < S; j++) {
                //Cộng lần lượt các giá trị màu B G R
                BGR[0] += (unsigned char)pPix[0];
                BGR[1] += (unsigned char)pPix[1];
                BGR[2] += (unsigned char)pPix[2];
                //Di chuyển pPix sang pixel tiếp theo
                pPix += nSrcByteInPix;
            }
            //Di chuyển pSrcRow sang dòng tiếp theo
            pSrcRow += nSrcByteInRow;
        }
        //Tính trung bình cộng cho từng giá trị màu
        //Ép kiểu lại thành unsigned char
        BGR[0] = (unsigned char)(BGR[0] / (S * S));
        BGR[1] = (unsigned char)(BGR[1] / (S * S));
        BGR[2] = (unsigned char)(BGR[2] / (S * S));
        //Gán lại cho p.B/G/R
        p.B = BGR[0];
        p.G = BGR[1];
        p.R = BGR[2];
        break;
    }
    //Trường hợp bpp == 32
    //Tương tự như case 24, chỉ khác ở chỗ ảnh 32bit sẽ có thêm biến Alpha
    //Nên ta sẽ gán, cộng và tính trung bình tương ứng cho 4 giá trị A B G R
```

Hình 6.2. Hàm tìm màu trung bình (theo từng ô vuông size S)

Để có thể thực hiện việc thu nhỏ ảnh trong hàm **zoomIn()** thì trước tiên ta phải tạo một hàm tính trung bình cộng giá trị màu trong 1 ô vuông có kích thước bằng tỉ lệ thu nhỏ **S** (hàm **findAveColor()**). Để thuận tiện cho việc tính toán và đơn giản hóa hàm, ta sẽ dùng hàm **unsigned char**. Trong hàm **findAveColor()**, ta xét các **case** khác nhau (**8, 24, 32**) để có thể tính toán một cách thích hợp (việc sử dụng switch case sẽ giảm thiểu thời gian chạy chương trình). Áp dụng thuật toán ở câu 3, ta sẽ tương tự xét các pixel trong ô vuông và nhảy một khoảng byte chính xác (nhờ **nSrcByteInPix** và **nSrcByteInRow**) để đạt được tổng giá trị màu có trong ô vuông đó. Tổng này sẽ được gán cho **aveColor** và được tính trung bình cộng, giá trị này sau đó sẽ được gán lại cho màu của ảnh (nếu là 24/32 bit) hoặc được trả lại dưới kiểu **unsigned char** (nếu là 8 bit)
Lí do biến **aveCol** và **BGR** phải được gán là **unsigned int** vì khi tính toán, giá trị có thể sẽ vượt quá kích thước của kiểu **char/unsigned char**, điều này sẽ gây ra việc tràn dữ liệu gây hỏng ảnh (việc có giá trị ban đầu của **BGR = {0}** cũng để tránh việc tràn dữ liệu). Khi gán kết quả lại cuối cùng, ta sẽ ép kiểu **unsigned char**.
Trường hợp 32 bit sẽ tương tự 24 bit nhưng lần này sẽ có thêm biến p.A (Alpha)

```
//Hàm scale ảnh xuống 1 tỉ lệ cho trước
int zoomIn(BMPImage srcBmp, BMPImage& desBmp, int S) {
```

Hình 7.1. Hàm thu nhỏ ảnh

```
//Đảm bảo compression = 0
desBmp.dib.compression = 0;
//Cho kích thước CŨ = trị tuyệt đối để tính toán
int oldHeight = (abs)(srcBmp.dib.height);
int oldWidth = (abs)(srcBmp.dib.width);

//Gán kích thước cho desBmp và floor (tránh trường hợp pixel lẻ)
desBmp.dib.height = (floor)(srcBmp.dib.height / S);
desBmp.dib.width = (floor)(srcBmp.dib.width / S);

//Cho kích thước MỚI = trị tuyệt đối để tính toán
int newHeight = (abs)(desBmp.dib.height);
int newWidth = (abs)(desBmp.dib.width);
```

Hình 7.2. Hàm thu nhỏ ảnh

Trong hàm **zoomIn(),** em có đảm bảo rằng phần **compression** sẽ bằng 0 để tránh đi trường hợp ảnh bị vỡ và lỗi màu. Lần lượt gán các giá trị cho **oldHeight**, **oldWidth**, **newHeight** và **newWidth** là trị tuyệt đối để phục vụ cho tính toán. Em còn cho **(floor) height** và **width** của ảnh **DESTINATION** để tránh khi chia theo tỉ lệ cho ra số dư.

```
//Trường hợp nếu resize ảnh 8bit ta cần tạo cho nó 1 bảng màu
if (desBmp.dib.bpp <= 8) {
    for (int i = 0; i < 256; i++) {
        desBmp.colorTable[i].B = i;
        desBmp.colorTable[i].G = i;
        desBmp.colorTable[i].R = i;
        desBmp.colorTable[i].Reserved = 0;
    }
}
```

Hình 7.3. Hàm thu nhỏ ảnh

Khi thao tác ảnh 8 bit, ta cần cấp lại một bảng màu mới (cấu trúc tương tự ở câu 3)

```
if (desBmp.dib.bpp == 8) {
    //Vì ảnh 8 bit chỉ có 1 giá trị màu nên ta chỉ gán lại 1 giá trị màu
    unsigned char ave = findAveColor(desBmp.dib, p, pPixSquare, nSrcByteInRow, nSrcByteInPix, S);
    pDesPix[0] = (char)ave;
}

if (desBmp.dib.bpp == 24) {
    //Ảnh 24 bit có 3 giá trị màu nên gán lại 3
    findAveColor(desBmp.dib, p, pPixSquare, nSrcByteInRow, nSrcByteInPix, S);
    pDesPix[0] = (char)p.B;
    pDesPix[1] = (char)p.G;
    pDesPix[2] = (char)p.R;
}
if (desBmp.dib.bpp == 32) {
    //Ảnh 32 bit có 4 giá trị màu nên gán lại 4
    findAveColor(desBmp.dib, p, pPixSquare, nSrcByteInRow, nSrcByteInPix, S);
    pDesPix[0] = (char)p.A;
    pDesPix[1] = (char)p.B;
    pDesPix[2] = (char)p.G;
    pDesPix[3] = (char)p.R;
}
```

Hình 7.4. Hàm thu nhỏ ảnh

Theo ý tưởng ban đầu, để có thể cho vòng lặp chạy theo từng ô vuông kích thước S và chạy hết các ô của ảnh **SOURCE** và **DESTINATION**, ta cần phải sử dụng 4 vòng lặp. Điều này sẽ làm cho code trở nên khó hiểu và rối, nên em đã tách vòng lặp chạy từng ô vuông kích thước S sang hàm **findAveColor()** (như đã nêu ở trên). Sau đó em có thể áp dụng lại cấu trúc của câu 3 cho lần lượt vòng lặp for chạy trong từng ô của ảnh **DESTINATION**, đồng thời gán giá trị tương ứng. 3 điều kiện if sẽ đảm bảo thao tác được thực hiên đúng trên định dạng file BMP đó.

```
//Hàm giải phóng bộ nhớ
void releaseMemory(BMPImage& srcImg, BMPImage& gray, BMPImage& zoom) {
    delete[] srcImg.pImageData, gray.pImageData, zoom.pImageData ;
    srcImg.pImageData = gray.pImageData = zoom.pImageData = NULL;
}
```

Hình 8. Hàm giải phóng bộ nhớ

Hàm giải phóng bộ nhớ của pImageData.

III. Hàm main:

```
if (argc > 5 || argc < 4) {
    cout << "So luong tham so khong hop le\n";
    cout << "Vui long thu lai\n";
}
```

Hình 9. Điều kiện kiểm tra số lượng tham số truyền vào

Điều kiện kiểm tra nếu số tham số truyền vào vượt/ít hơn yêu cầu.

```
if (argc == 4) {
    if (_strcmpi(argv[1], "-conv") != 0)
        cout << "Dong lenh chua dung!\n";
    else {
        readBMP(argv[2], srcImg);
        cout << "Thong tin anh SOURCE\n";
        printInfo(srcImg);
        to8Bit(srcImg, grayImg);
        writeBMP(argv[3], grayImg);
        printInfo(grayImg);
        cout << "Chuyen 8 bit thanh cong!\n";
    }

}
if (argc == 5) {
    if (_strcmpi(argv[1], "-zoom") != 0)
        cout << "Dong lenh chua dung!\n";
    else {
        S = atoi(argv[4]);
        readBMP(argv[2], srcImg);
        cout << "Thong tin anh SOURCE\n";
        printInfo(srcImg);
        zoomIn(srcImg, zoomImg, S);
        writeBMP(argv[3], zoomImg);
        printInfo(zoomImg);
        cout << "Thu nho anh thanh cong!\n";
    }

}
```

Hình 9. Điều kiện kiểm tra số tham số truyền vào để thực hiện chương trình

Kiểm tra số tham số được truyền vào, nếu là 4 thì sẽ thực hiện câu lệnh **-conv** theo cấu trúc
VD: 21120588.exe -conv D:\input.bmp D:\output.bmp
Kiểm tra số tham số được truyền vào, nếu là 5 thì sẽ thực hiện câu lệnh **-zoom** theo cấu trúc
VD: 21120588.exe -zoom D:\input.bmp D:\output.bmp 7

IV. Kết quả:

Sau bài tập lý thuyết 1, em đã thành công các thao tác đơn giản trên file BMP định dạng 8/24/32 bits thông qua việc vận dụng struct, vòng lặp for, switch case, con trỏ và lý thuyết về file BMP. Bài làm đã thực hiên đúng yêu cầu của thầy với số câu hoàn thiện là 5/5.

HẾT