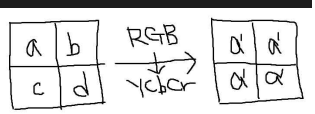
```
std::cin >> GetImage;
sprintf(ImagePath, "%s₩₩%s", ExePath, GetImage.c_str());
```

1. Image Load

std::cin을 이용하여 bmp 파일을 불러온다.

```
for (int y = 0; y < m_pImageLayer->m_Image.m_nH; ++y)
    for (int x = 0; x < m_pImageLayer->m_Image.m_nW; ++x)
    {
        InputY[y][x] = 0.29900 * m_pImageLayer->m_Image.m_Red[y][x] + 0.58700 * m_pImageLayer->m_Image.m_Green[y][x]
            + 0.11400 * m_pImageLayer->m_Image.m_Blue[y][x];
        InputCb[y][x] = -0.16874 * m_pImageLayer->m_Image.m_Green[y - y % 2][x - x % 2]
            - 0.33126 * m_pImageLayer->m_Image.m_Green[y - y % 2][x - x % 2]
            + 0.50000 * m_pImageLayer->m_Image.m_Blue[y - y % 2][x - x % 2];
        InputCr[y][x] = 0.50000 * m_pImageLayer->m_Image.m_Red[y - y % 2][x - x % 2]
            - 0.41869 * m_pImageLayer->m_Image.m_Green[y - y % 2][x - x % 2]
            - 0.08131 * m_pImageLayer->m_Image.m_Blue[y - y % 2][x - x % 2];
    }
```



2. RGB to YCbCr 변환

위의 그림과 같이 4픽셀 중 왼쪽 위의 픽셀을 기준으로 RGB to YCbCR 변환을 수행한 후 나머지 3픽셀을 전부 같은 값으로 변환해준다. 이를 전체에 대해서 실행한다.

이렇게 하게 되면 4픽셀의 정보를 1픽셀로 묶어서 데이터 정보량이 반으로 줄어든다.

y - y % 2 형태를 이용하여 이를 수행한다.

```
else if (bQuantization)
{
    DCT2D(InputY, OutY, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);
    DCT2D(InputCb, OutCb, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);
    DCT2D(InputCr, OutCr, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);

    int Y = 0;
    int Cb = 0;
    int Cr = 0;

    std::vector<int> vecYx;
    std::vector<int> vecYy;
    std::vector<int> vecCbx;
    std::vector<int> vecCby;
    std::vector<int> vecCrx;
    std::vector<int> vecCry;

    std::ofstream fout;
    fout.open("Histogram.txt");

    for (int y = 0; y < m_pImageLayer->m_Image.m_nH; ++y)
        for (int x = 0; x < m_pImageLayer->m_Image.m_nW; ++x)
        {
            OutY[y][x] = (std::max)((std::min)((int)(round(OutY[y][x] / InputY[y][x]) + 0.5), 255), 0);
            OutCb[y][x] = (std::max)((std::min)((int)(round(OutCb[y][x] / InputCb[y][x]) + 0.5), 255), 0);
            OutCr[y][x] = (std::max)((std::min)((int)(round(OutCr[y][x] / InputCr[y][x]) + 0.5), 255), 0);

            if ((int)OutY[y][x]) Y++;
            if ((int)OutCb[y][x]) Cb++;
            if ((int)OutCr[y][x]) Cr++;

            if (!x && !y) {
                vecYx.push_back((int)OutY[y][x]);
                vecYy.push_back(1);
                vecCbx.push_back((int)OutCb[y][x]);
                vecCby.push_back(1);
                vecCrx.push_back((int)OutCr[y][x]);
                vecCry.push_back(1);
            }
            else {

                auto findY = std::find(vecYx.begin(), vecYx.end(), (int)OutY[y][x]);
                auto findCb = std::find(vecCbx.begin(), vecCbx.end(), (int)OutCb[y][x]);
                auto findCr = std::find(vecCrx.begin(), vecCrx.end(), (int)OutCr[y][x]);

                if (findY == vecYx.end()) {
                    vecYx.push_back((int)OutY[y][x]);
                    vecYy.push_back(1);
                }
                else vecYy[findY - vecYx.begin()]++;

                if (findCb == vecCbx.end()) {
                    vecCbx.push_back((int)OutCb[y][x]);
                    vecCby.push_back(1);
                }
                else vecCby[findCb - vecCbx.begin()]++;

                if (findCr == vecCrx.end()) {
```

```
                vecCrx.push_back((int)OutCr[y][x]);
                vecCry.push_back(1);
            }
            else vecCry[findCr - vecCrx.begin()]++;
        }
    }

    fout << "₩n";
    for (int i = 0; i < vecYx.size(); i++) {
        fout << vecYx[i] << "₩t" << vecYy[i] << "₩n";
    }
    fout << "₩nCb₩n";

    for (int i = 0; i < vecCbx.size(); i++) {
        fout << vecCbx[i] << "₩t" << vecCby[i] << "₩n";
    }
    fout << "₩nCr₩n";

    for (int i = 0; i < vecCrx.size(); i++) {
        fout << vecCrx[i] << "₩t" << vecCry[i] << "₩n";
    }
    fout.close();

    std::cout << "Number Of 0 / Y : " << Y << ", Cb : " << Cb << ", Cr : " << Cr << "₩n";
}
```

3. DCT, Quantization

먼저 DCT 함수를 이용해서 Block Size를 8로 해서 DCT를 수행해준다.

다음으로 Quantization을 수행한다.

$\hat{c}(x,y) = ROUND(\frac{c(x,y)}{q(x,y)})$ 다음 수식을 적용하면 Quantization이 완료된다.

Divide By Zero 등의 문제가 발생할 수 있기 때문에 데이터를 가공해준다.

실제 계산에서는 데이터를 가공하지 않는다.

다음으로 Y Cb Cr 변수들을 이용하여 0의 개수를 센다.

다음으로 vecYx, vecYy, ... 를 이용하여 각각의 정보들이 몇 개 들어가 있는지를 계산하여 이를 히스토그램으로 변환할 수 있다.

```
else if (bInverseQuantization) {
    for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y)
        for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
        {
            OutY[y][x] = OutY[y][x] * InputY[y][x];
            OutCb[y][x] = OutCb[y][x] * InputCb[y][x];
            OutCr[y][x] = OutCr[y][x] * InputCr[y][x];
        }

    IDCT2D(OutY, InputY, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);
    IDCT2D(OutCb, InputCb, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);
    IDCT2D(OutCr, InputCr, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);

    for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y)
        for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
        {
            int R, G, B;

            R = 1.00000 * InputY[y][x] + 1.40200 * InputCr[y][x];
            G = 1.00000 * InputY[y][x] - 0.34414 * InputCb[y][x] - 0.71414 * InputCr[y][x];
            B = 1.00000 * InputY[y][x] + 1.77200 * InputCb[y][x];

            InputY[y][x] = R;
            InputCb[y][x] = G;
            InputCr[y][x] = B;
        }

    for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y)
        for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
        {
            m_pImageLayer->m_ImageOut.m_Red[y][x] = (std::max)((std::min)((int)(round(InputY[y][x]) + 0.5), 255), 0);
            m_pImageLayer->m_ImageOut.m_Green[y][x] = (std::max)((std::min)((int)(round(InputCb[y][x]) + 0.5), 255), 0);
            m_pImageLayer->m_ImageOut.m_Blue[y][x] = (std::max)((std::min)((int)(round(InputCr[y][x]) + 0.5), 255), 0);
        }
}
```

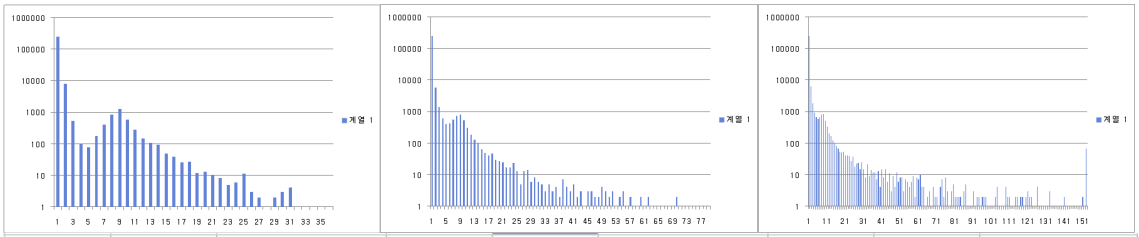
4. Inverse Quantization, IDCT, YCbCR to RGB 변환

위에서 표현했던 Quantization의 역연산을 수행하여 Inverse Quantization을 수행한다.

다음으로 IDCT 함수를 이용하여 IDCT를 수행한다.

다음으로 YCbCr to RGB 변환 공식을 이용하여 RGB로 표현한다.

화면에 출력할 때는 std::max, std::min 함수를 이용하여 값의 범위를 제한한다.



Y	Entropy =	0.116537369	Cb	Entropy =	0.125229417	Cr	Entropy =	0.155101306
0	249463	0.0204921022	0	249832	0.0199106388	0	247039	0.0242892545
1	7869	0.0457057943	1	5669	0.0360072125	1	6263	0.0387461368
2	533	0.00547308417	2	1392	0.0120798559	2	1847	0.0151629596
3	98	0.00128126815	3	606	0.00609381402	3	918	0.00859958674
4	78	0.00104928106	4	401	0.00430669256	4	670	0.00662594463
5	172	0.00208846273	5	425	0.00452352208	5	586	0.0059252792
6	406	0.00435205703	6	551	0.00562759768	6	652	0.0064773506
7	830	0.00791379287	7	724	0.00706700204	7	805	0.00771621305
8	1269	0.0112069487	8	796	0.00764477148	8	832	0.00792954486
9	572	0.00580663417	9	526	0.00541272544	9	499	0.00517844867
10	283	0.00320278639	10	309	0.00345204035	10	328	0.00363187589
11	149	0.00184462621	11	183	0.00220323279	11	201	0.00238870284
12	104	0.00134947466	12	130	0.00163878461	12	167	0.00203591353
13	94	0.00123546118	13	95	0.0012469389	13	122	0.00155077351
14	48	0.000684319827	14	63	0.000869787447	14	102	0.00132680456
15	38	0.000556460344	15	48	0.000684319827	15	81	0.00108457354
16	26	0.000397082256	16	40	0.000582348622	16	68	0.00093021434
17	27	0.00041066649	17	47	0.000671702488	17	54	0.00075932271
18	12	0.000198640076	18	30	0.00045105957	18	52	0.00073445092
19	13	0.000213469526	19	27	0.00041066649	19	53	0.00074690245
20	10	0.000168553922	20	25	0.000383434287	20	40	0.00058234862
21	8	0.000137800596	21	17	0.000271597088	21	41	0.0005952301
22	5	9.00187E-5	22	17	0.000271597088	22	38	0.00055646034
23	6	0.000106210068	23	23	0.000355936729	23	27	0.00041066649
24	11	0.000183672408	24	13	0.000213469526	24	37	0.00054345136
25	3	5.655E-5	25	5	9.00187E-5	25	18	0.00028586889
26	2	3.90435E-5	26	13	0.000213469526	26	22	0.00034208137
27	1	2.06701E-5	27	14	0.000228171411	27	23	0.00035593673
28	2	3.90435E-5	28	6	0.000106210068	28	15	0.00024275486

5. Histogram, Entropy

왼쪽부터 순서대로 Y Cb Cr 의 데이터 도수분포표이다.

$H(P) = -\sum_{x} P(x)\log P(x)$ 각 채널의 모든 값에 대해 다음 수식을 적용하면 Entropy를 구할 수 있다. 가장 상위에 있는 값이 채널의 Entropy 값이다.

6. 영상별 PSNR

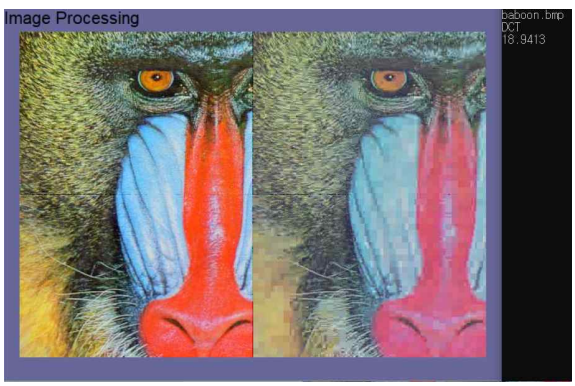



Image Processing
bird.bmp
DCT
18.7876
Image Processing
camera.bmp
DCT
19.9798
Image Processing
couple.bmp
DCT
20.228
Image Processing
girl.bmp
DCT
14.5981


Image Processing
house.bmp
DCT
22.1862