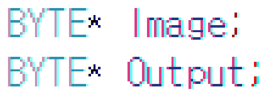
Color ➔ 24 bits

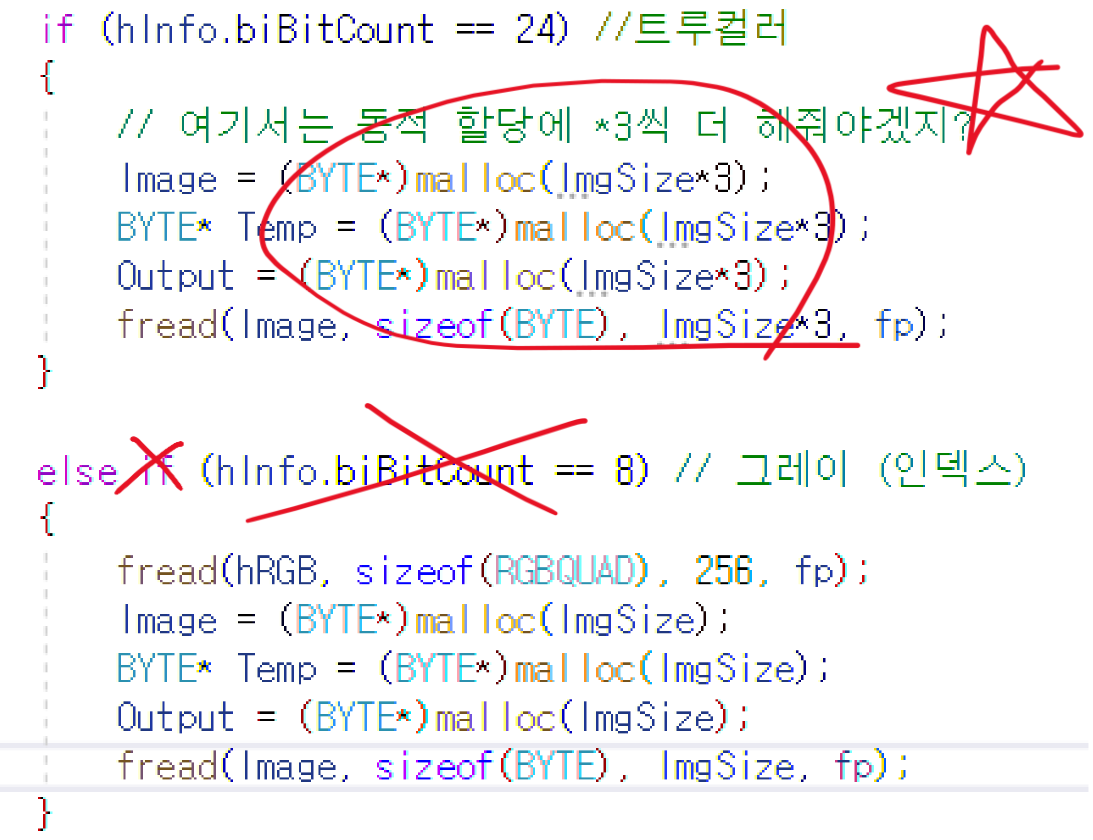
- -8 bits
- 14
- 40
- 1024
- 영상 픽셀 정보(1픽셀 = 8bits)
- **8bit -> 256 level
- **하나의 픽셀당 8bits (1byte)

- -24 bits True color(약 1600만 color = 256*256*256)
- 14
- 40
- ---팔레트 필요 X
- 영상 픽셀 정보(1픽셀 = 24bits)

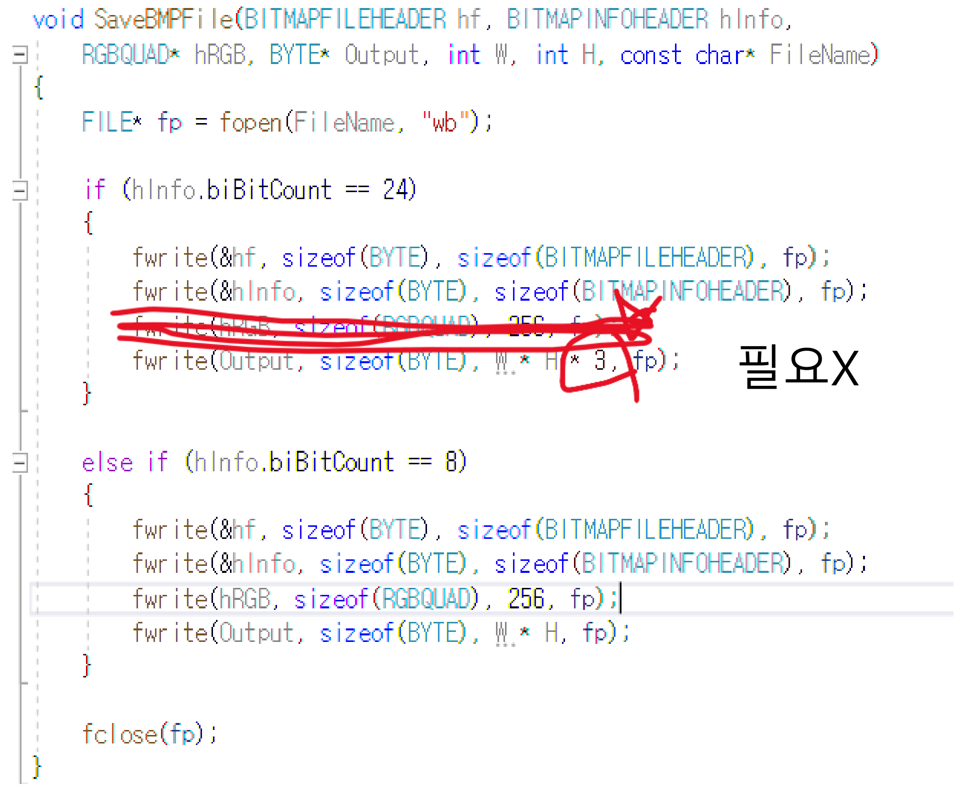


```
BYTE* Image;
BYTE* Output;
```

If 문 안에서 변수 지정하면
그 안에서만 인식 가능

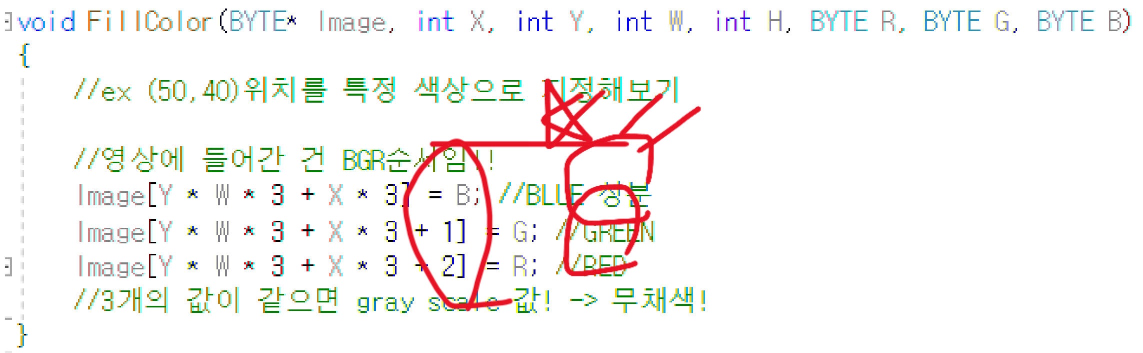


```
if (hInfo.biBitCount == 24) //트루컬러
{
    // 여기서는 동적 할당에 *3씩 더 해줘야겠지?
    Image = (BYTE*)malloc(ImgSize*3);
    BYTE* Temp = (BYTE*)malloc(ImgSize*3);
    Output = (BYTE*)malloc(ImgSize*3);
    fread(Image, sizeof(BYTE), ImgSize*3, fp);
}

else if (hInfo.biBitCount == 8) // 그레이 (인덱스)
{
    fread(hRGB, sizeof(RGBQUAD), 256, fp);
    Image = (BYTE*)malloc(ImgSize);
    BYTE* Temp = (BYTE*)malloc(ImgSize);
    Output = (BYTE*)malloc(ImgSize);
    fread(Image, sizeof(BYTE), ImgSize, fp);
}
```

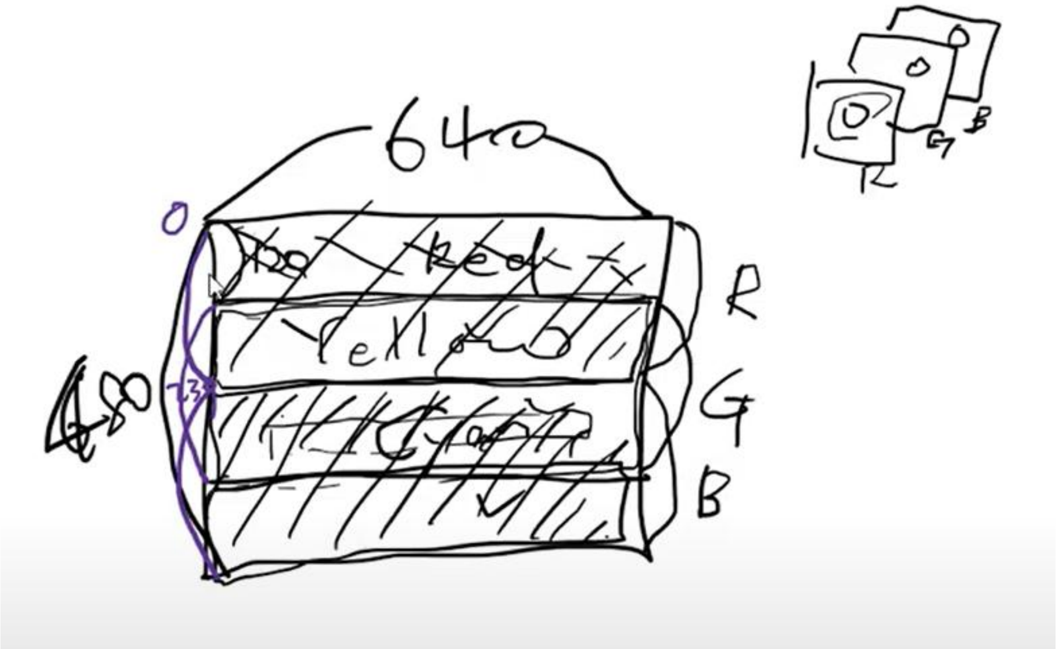


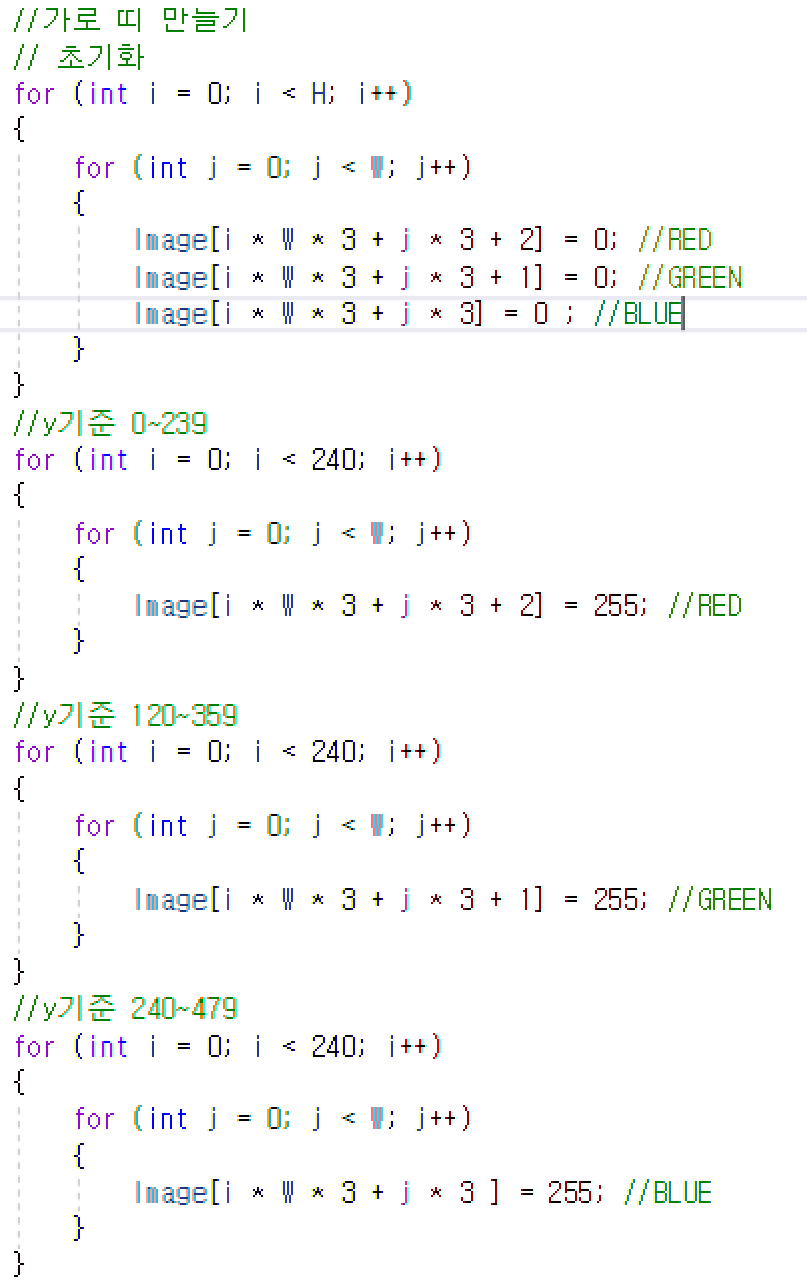
```
void SaveBMPFile(BITMAPFILEHEADER hf, BITMAPINFOHEADER hInfo,
    RGBQUAD* hRGB, BYTE* Output, int W, int H, const char* FileName)
{
    FILE* fp = fopen(FileName, "wb");

    if (hInfo.biBitCount == 24)
    {
        fwrite(&hf, sizeof(BYTE), sizeof(BITMAPFILEHEADER), fp);
        fwrite(&hInfo, sizeof(BYTE), sizeof(BITMAPINFOHEADER), fp);
        ~~fwrite(hRGB, sizeof(RGBQUAD), 256, fp);~~
        fwrite(Output, sizeof(BYTE), W * H * 3, fp);
    }

    else if (hInfo.biBitCount == 8)
    {
        fwrite(&hf, sizeof(BYTE), sizeof(BITMAPFILEHEADER), fp);
        fwrite(&hInfo, sizeof(BYTE), sizeof(BITMAPINFOHEADER), fp);
        fwrite(hRGB, sizeof(RGBQUAD), 256, fp);
        fwrite(Output, sizeof(BYTE), W * H, fp);
    }

    fclose(fp);
}
```

필요X



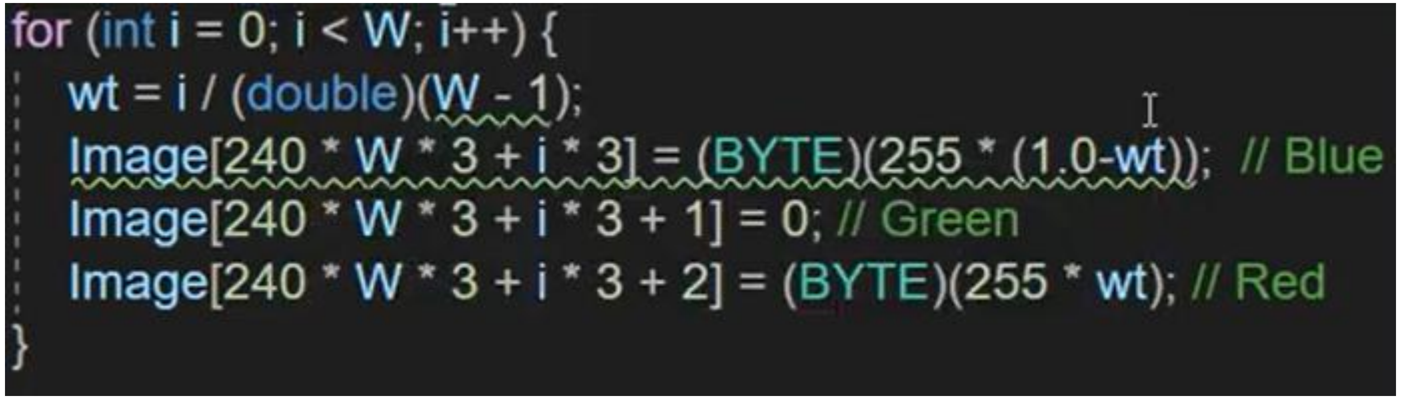
```
void FillColor(BYTE* Image, int X, int Y, int W, int H, BYTE R, BYTE G, BYTE B)
{
    //ex (50,40)위치를 특정 색상으로 지정해보기

    //영상에 들어간 건 BGR순서임!!
    Image[Y * W * 3 + X * 3] = B; //BLUE 성분
    Image[Y * W * 3 + X * 3 + 1] = G; //GREEN
    Image[Y * W * 3 + X * 3 + 2] = R; //RED
    //3개의 값이 같으면 gray scale값! -> 무채색!
}
```



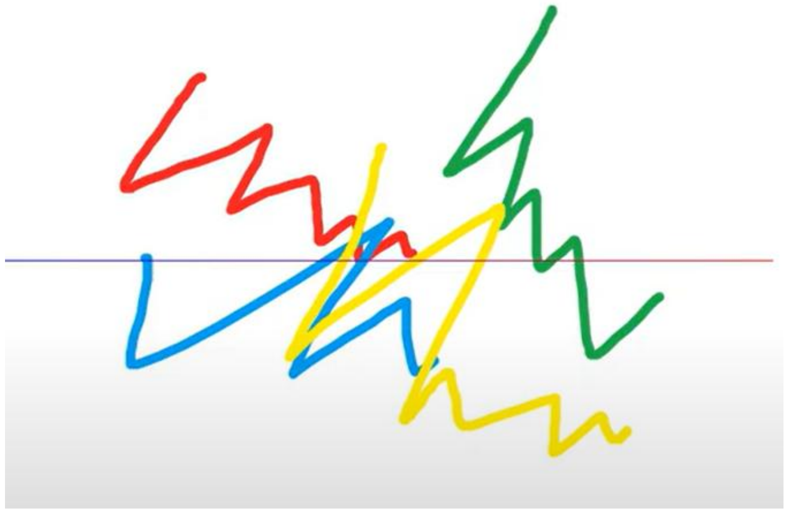


```
//가로 띠 만들기
// 초기화
for (int i = 0; i < H; i++)
{
    for (int j = 0; j < W; j++)
    {
        Image[i * W * 3 + j * 3 + 2] = 0; //RED
        Image[i * W * 3 + j * 3 + 1] = 0; //GREEN
        Image[i * W * 3 + j * 3] = 0 ; //BLUE
    }
}
//y기준 0~239
for (int i = 0; i < 240; i++)
{
    for (int j = 0; j < W; j++)
    {
        Image[i * W * 3 + j * 3 + 2] = 255; //RED
    }
}
//y기준 120~359
for (int i = 0; i < 240; i++)
{
    for (int j = 0; j < W; j++)
    {
        Image[i * W * 3 + j * 3 + 1] = 255; //GREEN
    }
}
//y기준 240~479
for (int i = 0; i < 240; i++)
{
    for (int j = 0; j < W; j++)
    {
        Image[i * W * 3 + j * 3 ] = 255; //BLUE
    }
}
```

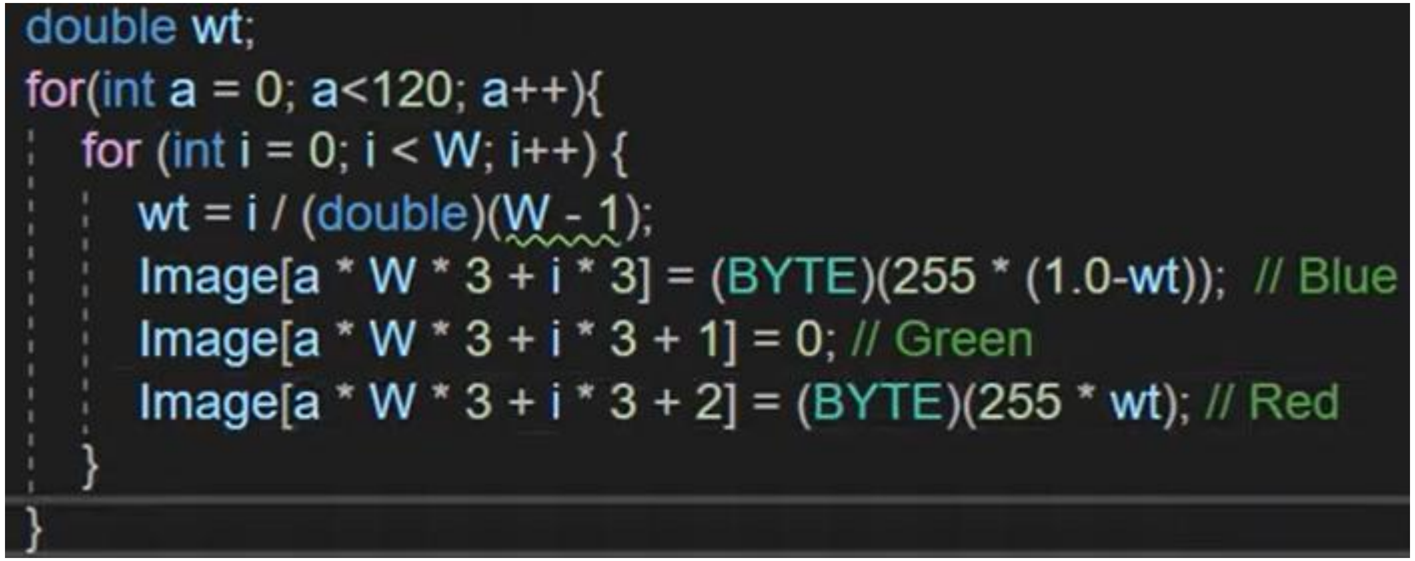
라인



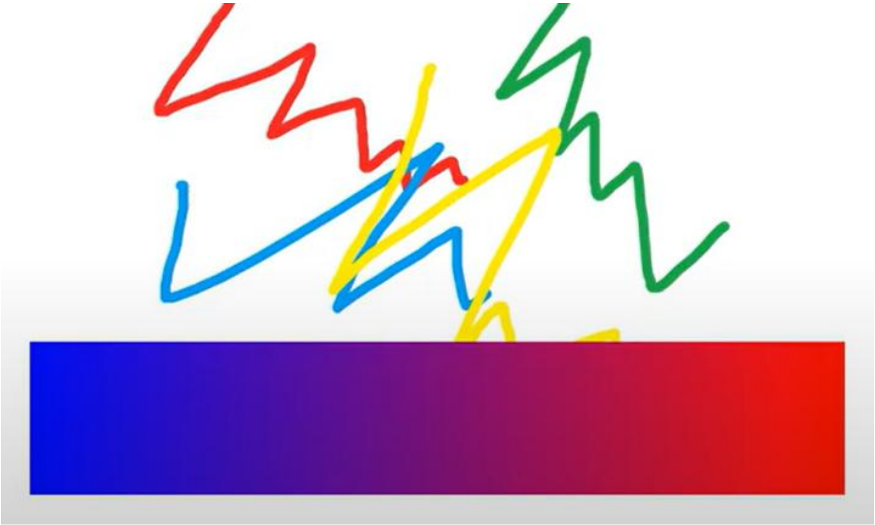
```
for (int i = 0; i < W; i++) {
    wt = i / (double)(W - 1);
    Image[240 * W * 3 + i * 3] = (BYTE)(255 * (1.0-wt));  // Blue
    Image[240 * W * 3 + i * 3 + 1] = 0; // Green
    Image[240 * W * 3 + i * 3 + 2] = (BYTE)(255 * wt); // Red
}
```



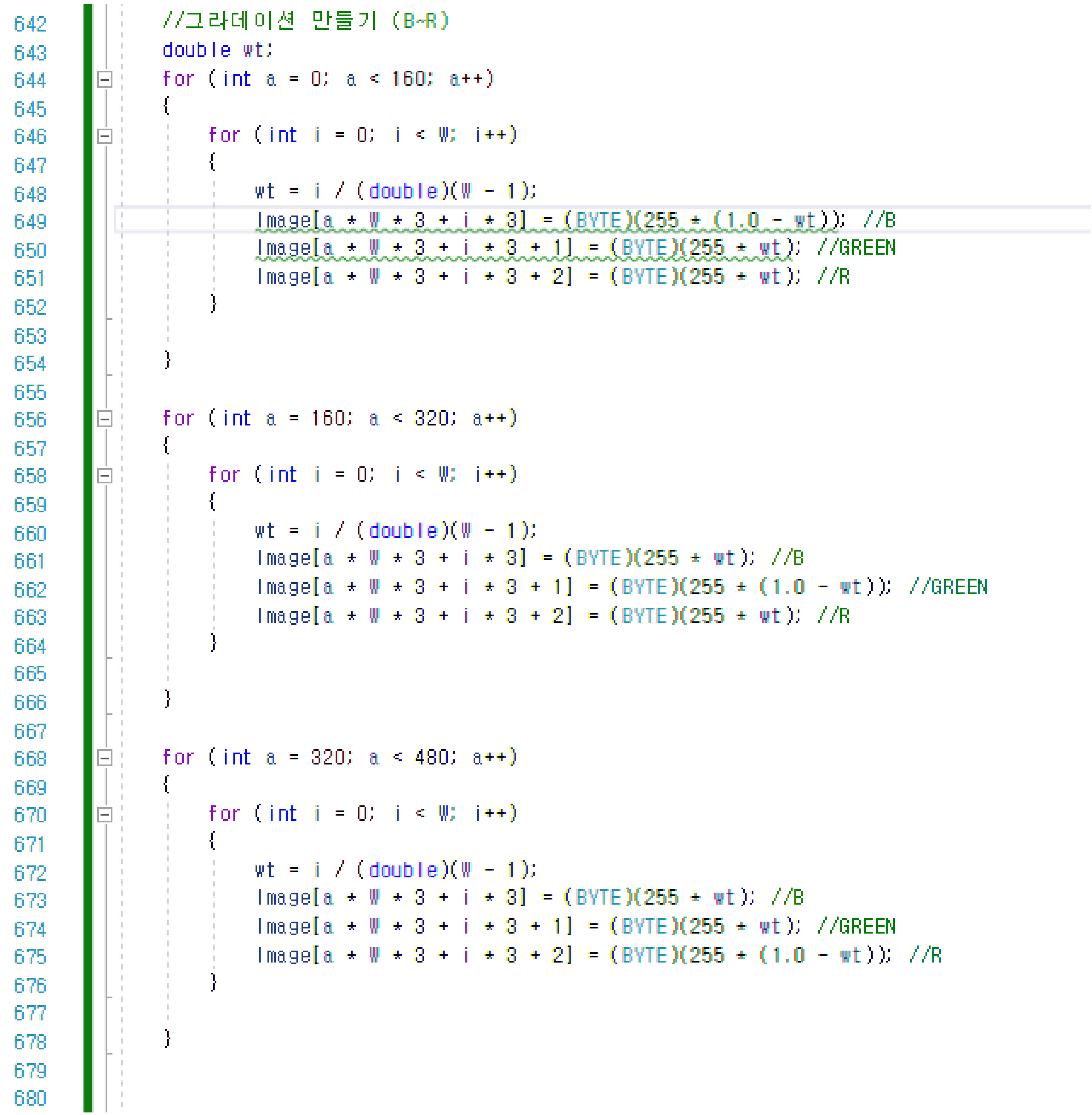
채우기 (사각형)



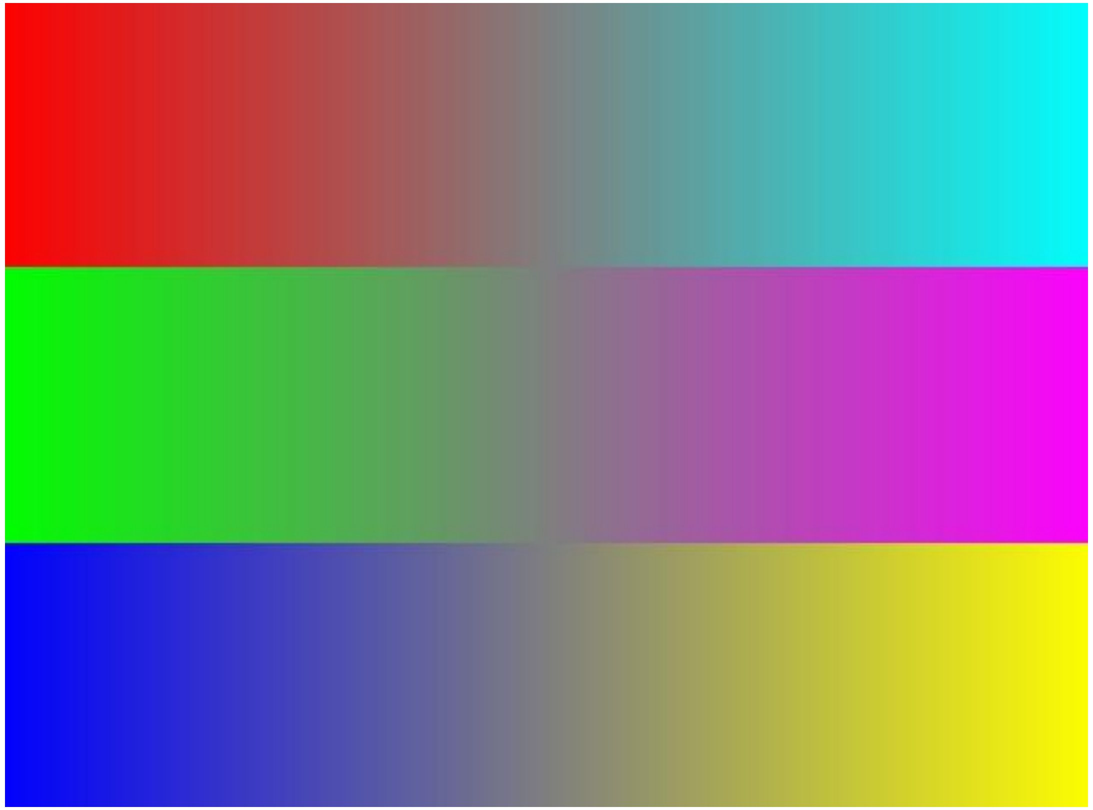
```
double wt;
for(int a = 0; a<120; a++){
    for (int i = 0; i < W; i++) {
        wt = i / (double)(W - 1);
        Image[a * W * 3 + i * 3] = (BYTE)(255 * (1.0-wt));  // Blue
        Image[a * W * 3 + i * 3 + 1] = 0; // Green
        Image[a * W * 3 + i * 3 + 2] = (BYTE)(255 * wt); // Red
    }
}
```



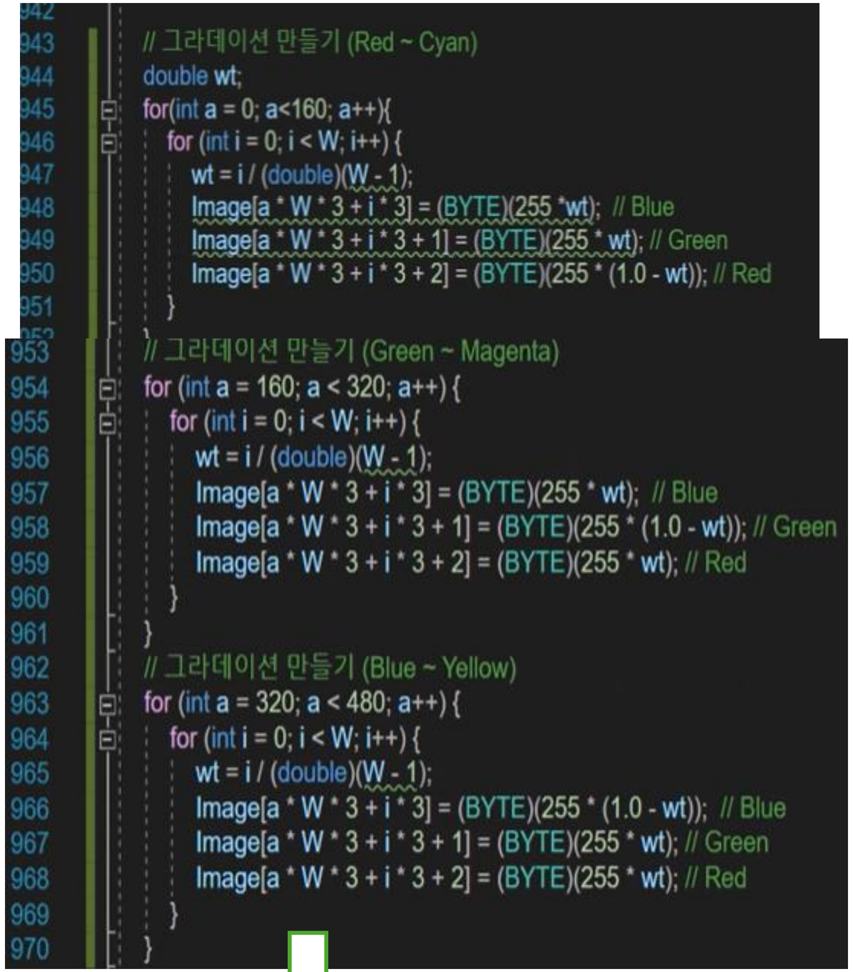
10주차 과제



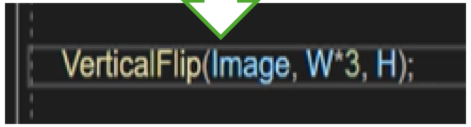
```
//그라데이션 만들기 (B~R)
double wt;
for (int a = 0; a < 160; a++)
{
    for (int i = 0; i < W; i++)
    {
        wt = i / (double)(W - 1);
        Image[a * W * 3 + i * 3] = (BYTE)(255 * (1.0 - wt)); //B
        Image[a * W * 3 + i * 3 + 1] = (BYTE)(255 * wt); //GREEN
        Image[a * W * 3 + i * 3 + 2] = (BYTE)(255 * wt); //R
    }

}

for (int a = 160; a < 320; a++)
{
    for (int i = 0; i < W; i++)
    {
        wt = i / (double)(W - 1);
        Image[a * W * 3 + i * 3] = (BYTE)(255 * wt); //B
        Image[a * W * 3 + i * 3 + 1] = (BYTE)(255 * (1.0 - wt)); //GREEN
        Image[a * W * 3 + i * 3 + 2] = (BYTE)(255 * wt); //R
    }

}

for (int a = 320; a < 480; a++)
{
    for (int i = 0; i < W; i++)
    {
        wt = i / (double)(W - 1);
        Image[a * W * 3 + i * 3] = (BYTE)(255 * wt); //B
        Image[a * W * 3 + i * 3 + 1] = (BYTE)(255 * wt); //GREEN
        Image[a * W * 3 + i * 3 + 2] = (BYTE)(255 * (1.0 - wt)); //R
    }

}
```



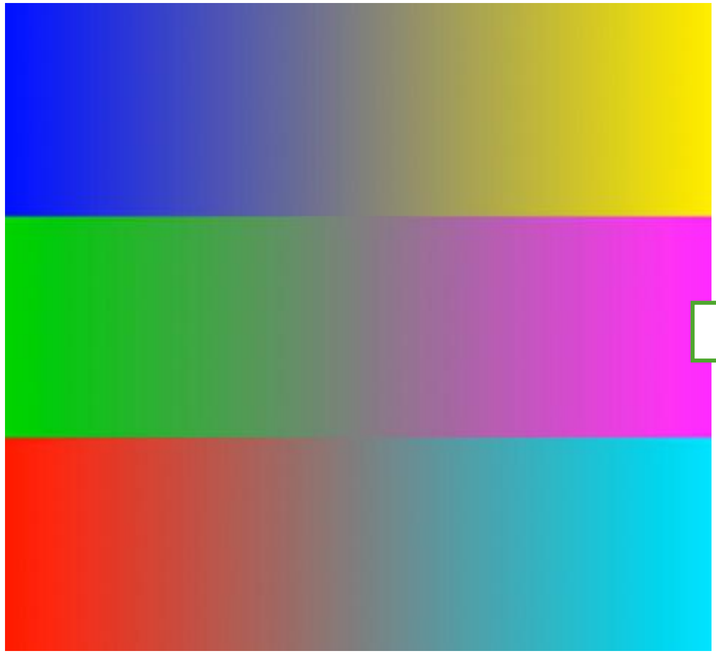
교수님 코드

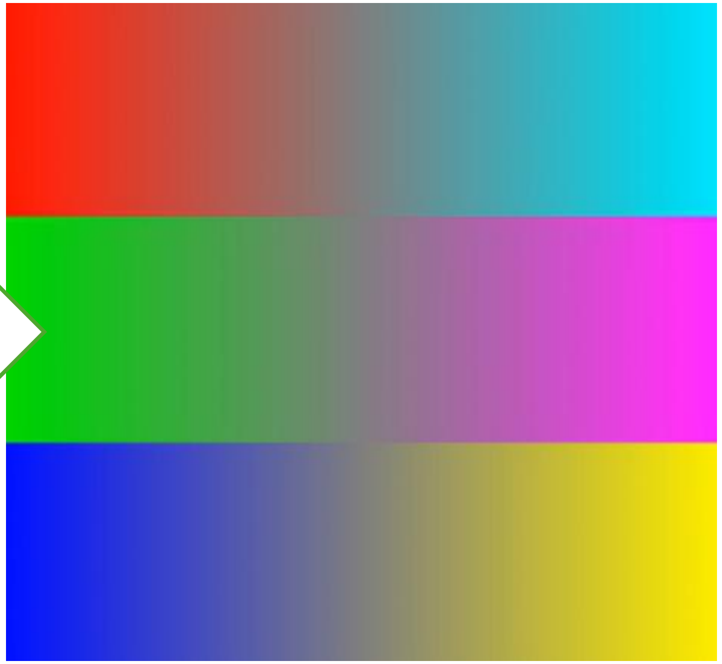


```
// 그라데이션 만들기 (Red ~ Cyan)
double wt;
for(int a = 0; a<160; a++){
    for (int i = 0; i < W; i++) {
        wt = i / (double)(W - 1);
        Image[a * W * 3 + i * 3] = (BYTE)(255 *wt);  // Blue
        Image[a * W * 3 + i * 3 + 1] = (BYTE)(255 * wt); // Green
        Image[a * W * 3 + i * 3 + 2] = (BYTE)(255 * (1.0 - wt)); // Red
    }
}
// 그라데이션 만들기 (Green ~ Magenta)
for (int a = 160; a < 320; a++) {
    for (int i = 0; i < W; i++) {
        wt = i / (double)(W - 1);
        Image[a * W * 3 + i * 3] = (BYTE)(255 * wt);  // Blue
        Image[a * W * 3 + i * 3 + 1] = (BYTE)(255 * (1.0 - wt)); // Green
        Image[a * W * 3 + i * 3 + 2] = (BYTE)(255 * wt); // Red
    }
}
// 그라데이션 만들기 (Blue ~ Yellow)
for (int a = 320; a < 480; a++) {
    for (int i = 0; i < W; i++) {
        wt = i / (double)(W - 1);
        Image[a * W * 3 + i * 3] = (BYTE)(255 * (1.0 - wt));  // Blue
        Image[a * W * 3 + i * 3 + 1] = (BYTE)(255 * wt); // Green
        Image[a * W * 3 + i * 3 + 2] = (BYTE)(255 * wt); // Red
    }
}
```



```
VerticalFlip(Image, W*3, H);
```





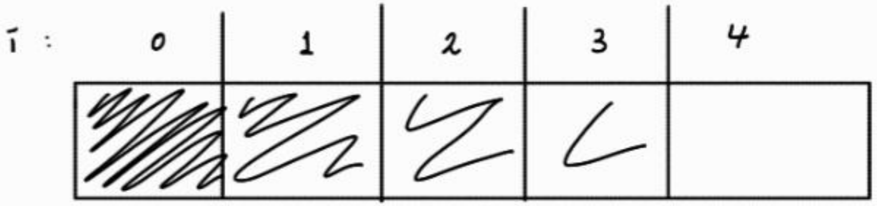
wt = i / (w-1) → 오른쪽으로 갈수록 숫자 大

⇒ 광약력 ↑

w = 5

w 크기

★ BMP는 반대 ★



wt :

0 / 4	1 /4	2/4	3/4	4/4
= 0	= 0.25	= 0.5	= 0.75	= 1

→ 값이 조금씩 커진다

색 배치

B = 255 * (1-wt)
G = 255 * (1-wt)
R = 255 * wt

가능?

B : 255
G : 255
R : 0

B : 191 ← 255 * 0.75
G : 191 ← 255 * 0.75
R : 63 ← 255 * 0.25

B ~~255 * 0~~ → 0
G ~~255 * 0~~ → 0
R 255 * 1 = 255

Y —— 회색 ——→ R

BMP 정리

1. * wt ⇒ 우측값 영역표시
   * (1-wt) ⇒ 좌측값 영역표시

```
2.  for ( int   a =        ; a =        ;  a++ )
                세로영역 지정 (행)
       for ( int   i=        ;  i =        ;  i++ )
                가로 영역지정 (열)
         {
                  (행)                     (열)
           Image [ a * w * 3  +  i * 3 ] ⇒ B
                 [             + i * 3 + 1 ] ⇒ G
                 [             + i * 3 + 2 ] ⇒ R
         }
```