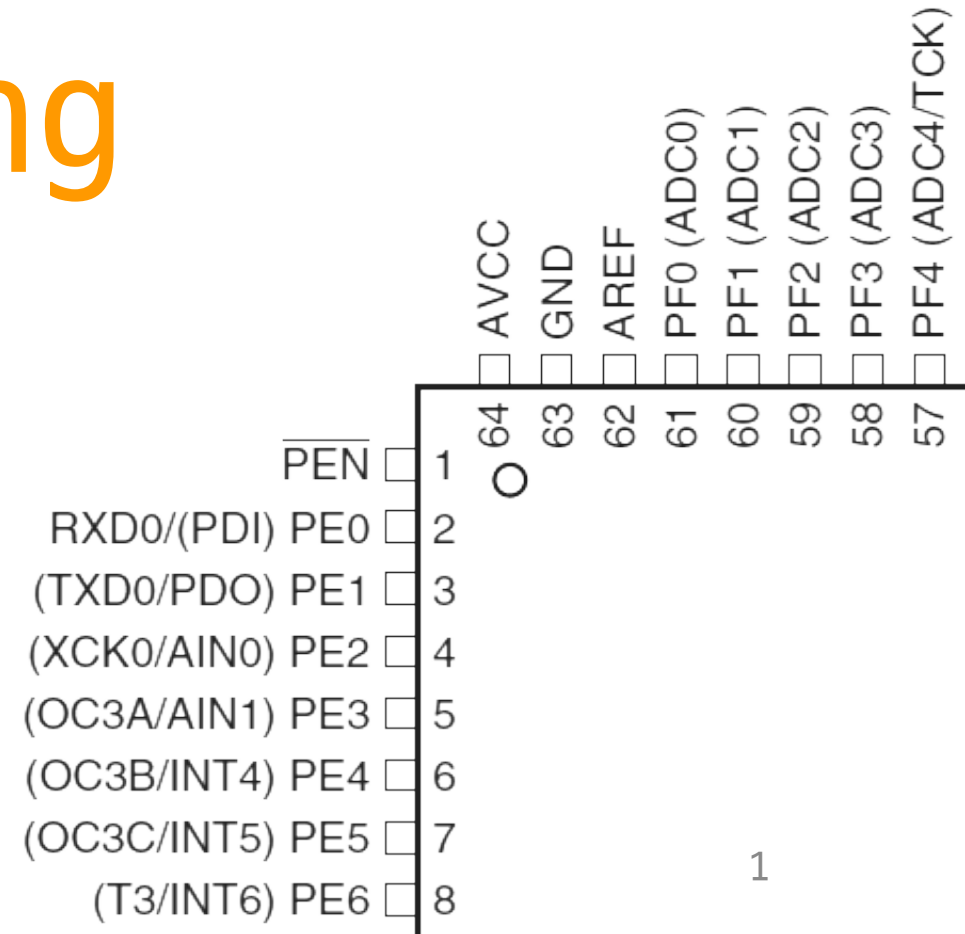
BMP data processing



• BMP

➢ 그림을 저장하는 데이터 규격

- ➢ 각 픽셀당 1 ~ 32비트의 색상을 저장할 수 있음.
  - ➢ BPP라고 함 : Bit per Pixel
  - ➢ 1~8 BPP의 경우 "팔레트"를 사용하여 영상을 저장
    - ➢ 각 픽셀에는 팔레트 번호 값만 지정
    - ➢ 팔레트에 실제 출력하는 색상 값이 저장되어 있음.
      - ➢ ex : (100, 121) 픽셀에 "37"이 저장된 경우, 37번 팔레트에 있는 색을 출력함.
      - ➢ 만약 37번 팔레트 색상이 노란 색인 경우, (100, 121) 픽셀은 노란색을 출력.
  - ➢ 16 ~ 32 BPP의 경우, 각 픽셀에 빛의 삼원색(Blue, Green, Red)의 값을 저장함.
    - ➢ 일반적으로 24BPP를 사용(각 채널당 8비트 할당)
      - ➢ B, G, R 각 채널별로 2^8 = 256가지의 색상을 출력 가능하고 3개 채널을 조합하여 색을 출력함.
        - 2^8 * 2^8 * 2^8 = 16,777,216 가지의 색상을 표현 가능

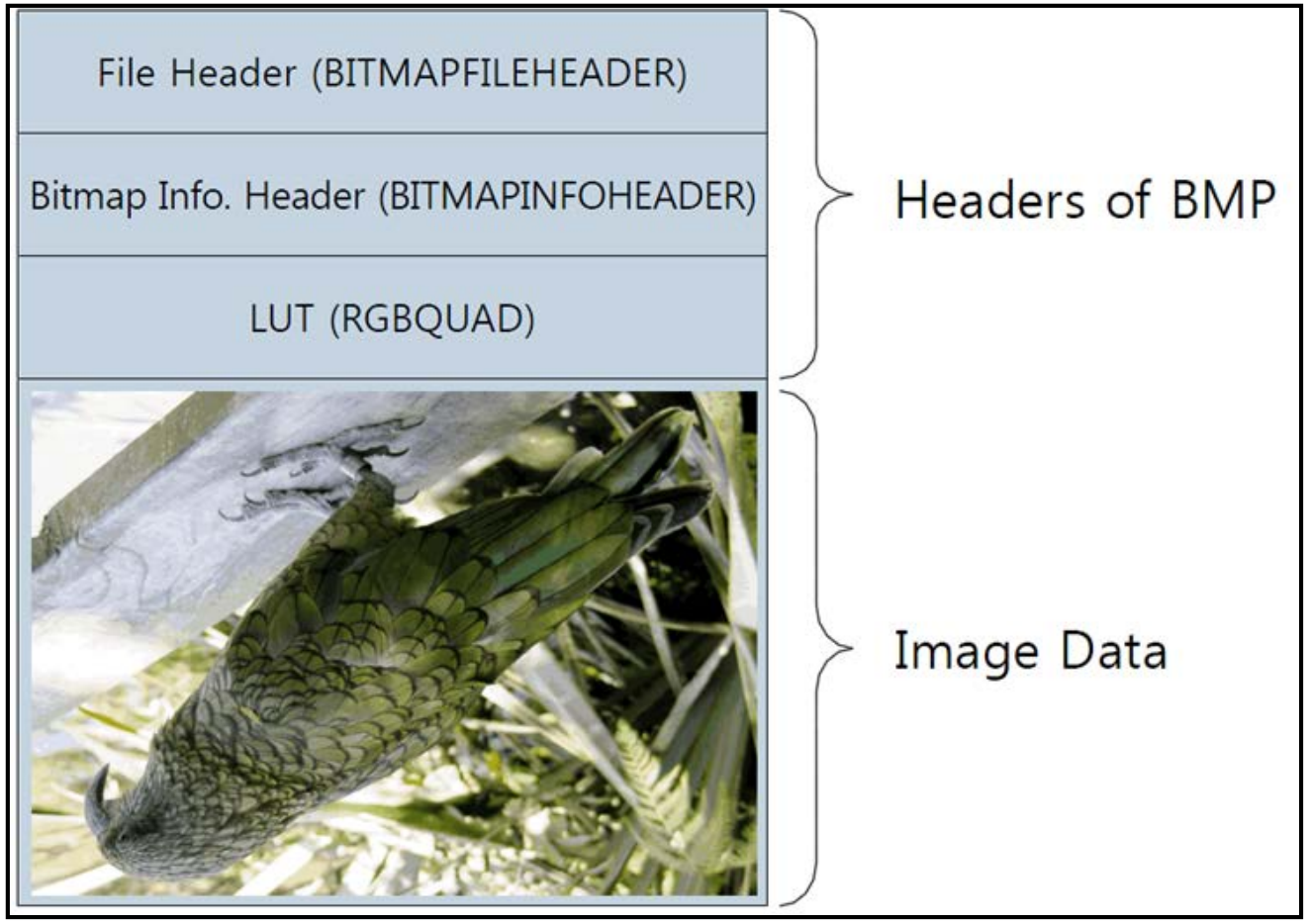
•BMP 파일의 구조

➢헤더 + 데이터

➢헤더 : 비트맵 파일의 정보를 저장하고 있음.

➢LUT : Look Up Table, 팔레트를 의미한다. BPP가 8을 초과하는 경우, LUT는 비트맵에 없을 수 있다.

➢데이터 : 실제 출력되는 데이터를 저장함. 우리가 생각하는 그림을 위아래를 뒤집은 순서로 저장함.



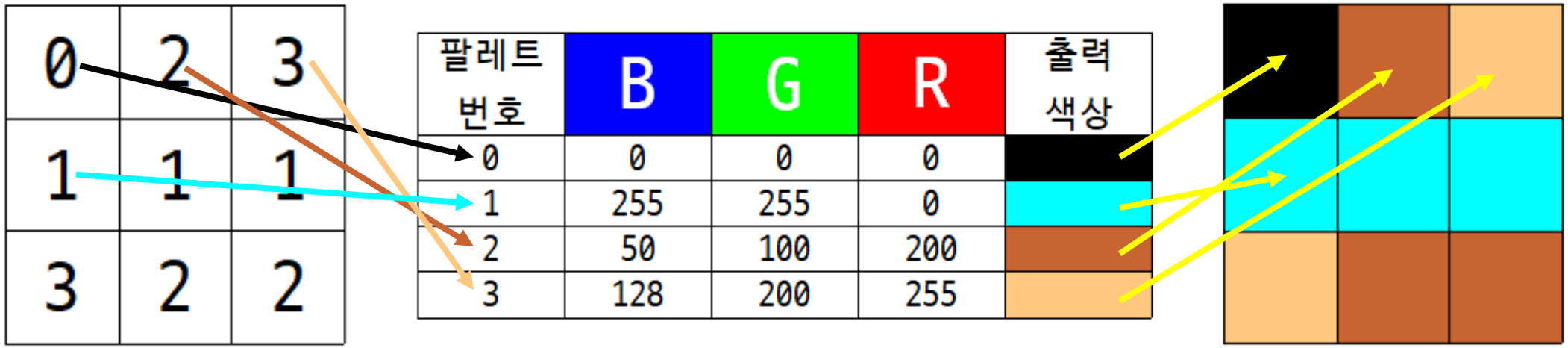
•팔레트의 활용(1 ~ 8 BPP의 경우)

➢각 픽셀에는 실제 출력되는 색상이 아니라 팔레트 번호가 저장됨.

➢따라서 응용프로그램은 팔레트를 먼저 읽고, 팔레트 번호에 따른 색상을 출력함.

➢만약 팔레트에 저장된 색이 바뀌면?

➢출력되는 색상도 바뀐다! -> palette swap!!



각 픽셀에 저장된 팔레트 번호

팔레트에 저장된 색상

실제 출력되는 색상

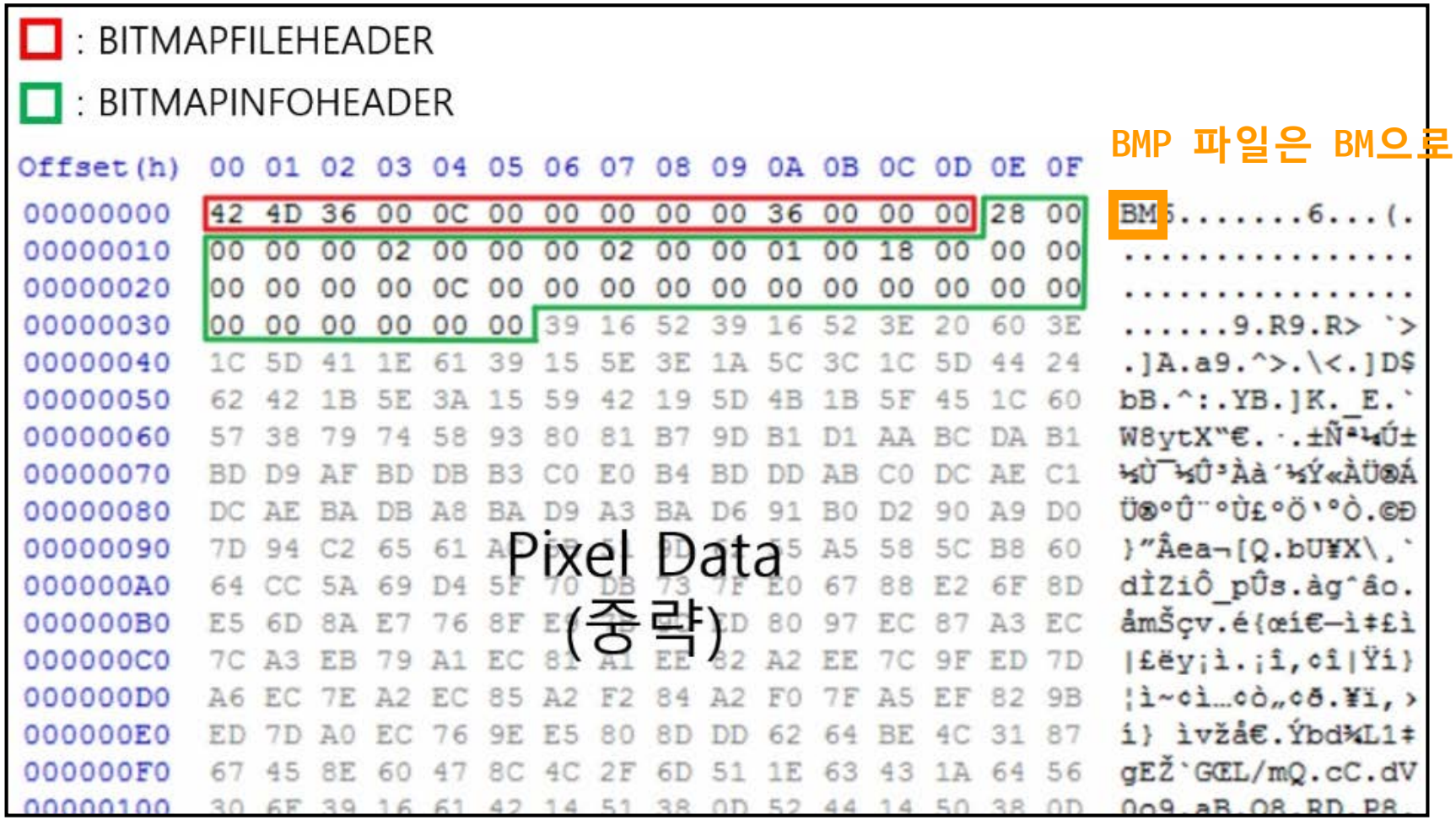
•파일 헤더(1)

```
typedef struct tagBITMAPFILEHEADER {
        WORD    bfType;   // 비트맵 파일을 알리는 문자열 "BM"
        DWORD   bfSize;   // 비트맵 파일의 크기
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits; // 실제 데이터가 시작되는 오프셋
} BITMAPFILEHEADER, FAR *LPBITMAPFILEHEADER,
*PBITMAPFILEHEADER;
```

• 파일 헤더(2)

```
typedef struct tagBITMAPINFOHEADER{
        DWORD       biSize;             // 구조체의 크기
        LONG        biWidth;            // 비트맵의 가로 길이(Pixel)
        LONG        biHeight;           // 비트맵의 세로 길이(Pixel)
        WORD        biPlanes;           // 비트 플레인 수(항상 1로 고정)
        WORD        biBitCount;         // 픽셀당 비트 수(BPP)
        DWORD       biCompression;      // 압축 유형
        DWORD       biSizeImage;        // 비트맵 데이터의 크기(압축 풀 때 사용)
        LONG        biXPelsPerMeter;    // 수평 해상도 (Pixel/meter)
        LONG        biYPelsPerMeter;    // 수직 해상도 (Pixel/meter)
        DWORD       biClrUsed;          // LUT에 포함된 컬러 인덱스 개수
        DWORD       biClrImportant;     // 사용한 컬러 인덱스 개수
} BITMAPINFOHEADER, FAR *LPBITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

•BMP 파일 구조 확인


: BITMAPFILEHEADER
: BITMAPINFOHEADER
BMP 파일은 BM으로 시작한다
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 42 4D 36 00 0C 00 00 00 00 00 36 00 00 00 28 00
00000010 00 00 00 02 00 00 00 02 00 00 01 00 18 00 00 00
00000020 00 00 00 00 0C 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 39 16 52 39 16 52 3E 20 60 3E
00000040 1C 5D 41 1E 61 39 15 5E 3E 1A 5C 3C 1C 5D 44 24
00000050 62 42 1B 5E 3A 15 59 42 19 5D 4B 1B 5F 45 1C 60
00000060 57 38 79 74 58 93 80 81 B7 9D B1 D1 AA BC DA B1
00000070 BD D9 AF BD DB B3 C0 E0 B4 BD DD AB C0 DC AE C1
00000080 DC AE BA DB A8 BA D9 A3 BA D6 91 B0 D2 90 A9 D0
00000090 7D 94 C2 65 61
000000A0 64 CC 5A 69 D4
000000B0 E5 6D 8A E7 76
000000C0 7C A3 EB 79 A1
000000D0 A6 EC 7E A2 EC 85 A2 F2 84 A2 F0 7F A5 EF 82 9B
000000E0 ED 7D A0 EC 76 9E E5 80 8D DD 62 64 BE 4C 31 87
000000F0 67 45 8E 60 47 8C 4C 2F 6D 51 1E 63 43 1A 64 56
00000100
Pixel Data
(중략)

•팔레트 구조체

```
typedef struct tagRGBQUAD {
        BYTE    rgbBlue;        // 파란색 값
        BYTE    rgbGreen;       // 녹색 값
        BYTE    rgbRed;         // 빨간색 값
        BYTE    rgbReserved;    // Reserved
} RGBQUAD;
```

* 참고 : RGBQUAD는 windows에서 사용하는 팔레트 구조체이며, OS/2에서는 다른 팔레트 구조체를 사용한다.

• 예제 실행

- 배포한 C파일에서(5_TH_C_src.c) 아래 코드의 주석을 함수별로 지우면서 예제를 실행하고 그 결과를 확인합니다
  - 한번에 한 함수만 실행시키세요
  - 소스코드를 수정해서 다른 bmp파일을 입력해서 수행해보세요. (paletteswap() 예제 제외)

```
int main()
{
    //bitof24_to_8bit_gray();
    //rgbdis();
    //crop();
    //paletteswap();
    //egdt();

    return 0;
}
```

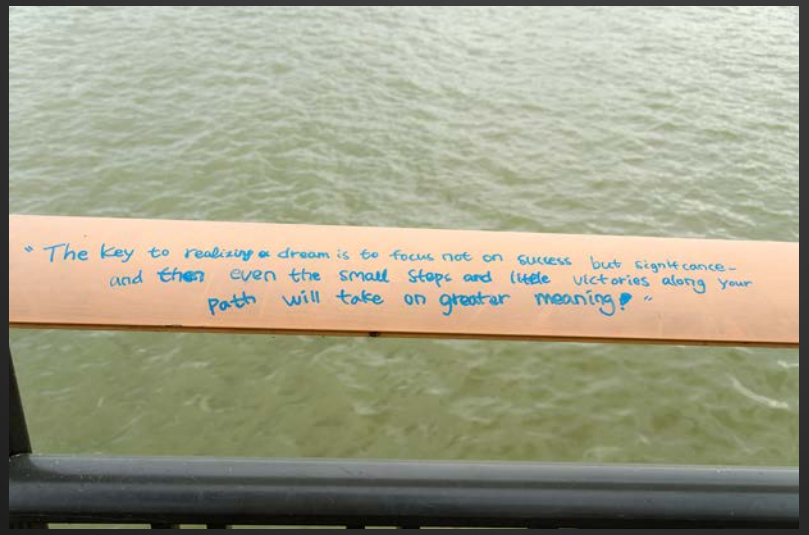
•예제 1 - bitof24_to_8bit_gray()

➢함께 배포한 ori.bmp파일을 흑백으로 변경하여 저장합니다.

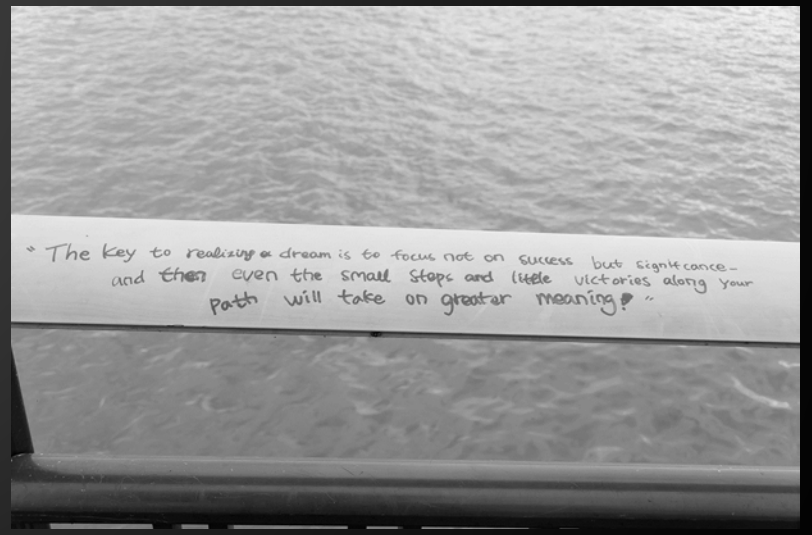
➢레포트에 "가로방향 데이터 크기는 4의 배수(바이트단위)"로 정한 이유를 찾아서 작성하세요.

➢GRAY = (BYTE)(0.299*R + 0.587*G + 0.114*B);에서 각 계수들이 어떻게 정해진 것인지 찾아서 레포트에 반영하세요.

```
RUN bitof24_to_8bit_gray()

File size : 3015992
offset : 54
Image Size : (1232X816)
24 BPP
Check BW.bmp!!
```



ori.bmp

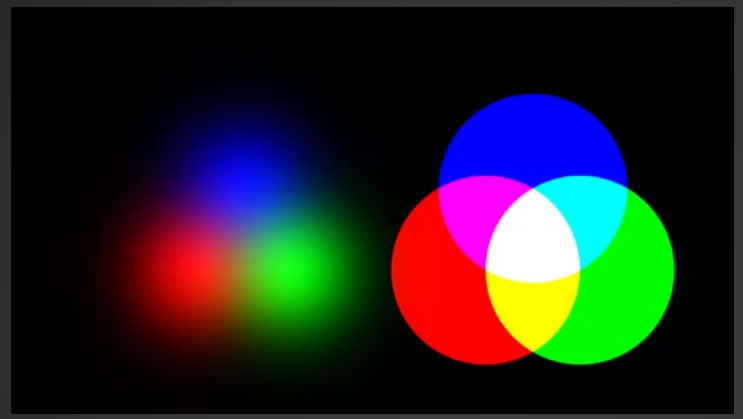


BW.bmp

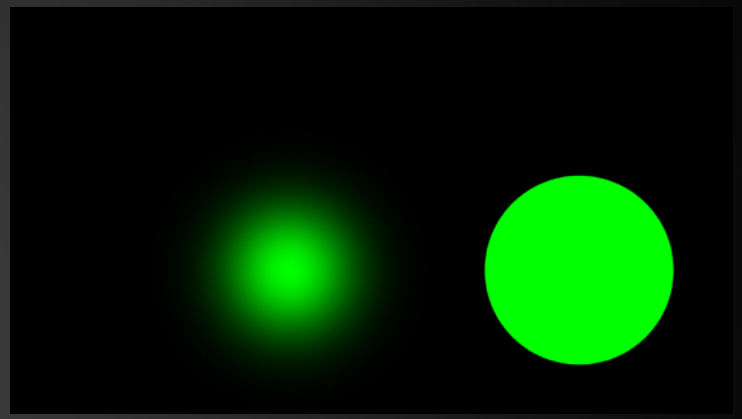
• 예제 2 - rgbdis()

➢ 함께 배포한 rgbdis.bmp파일의 내용을 B, G, R 채널별로 나누어 저장합니다.

```
RUN rgbdis()

File size : 6220856
offset : 54
Image Size : (1920X1080)
24 BPP
Check blue.bmp, green.bmp, red.bmp!!!
```

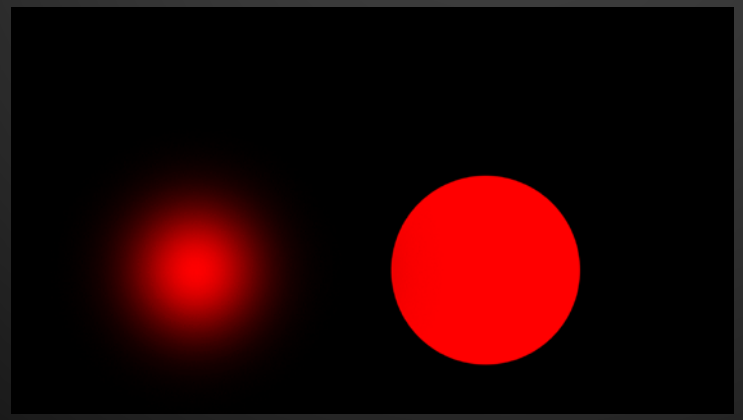


rgbdis.bmp ↑

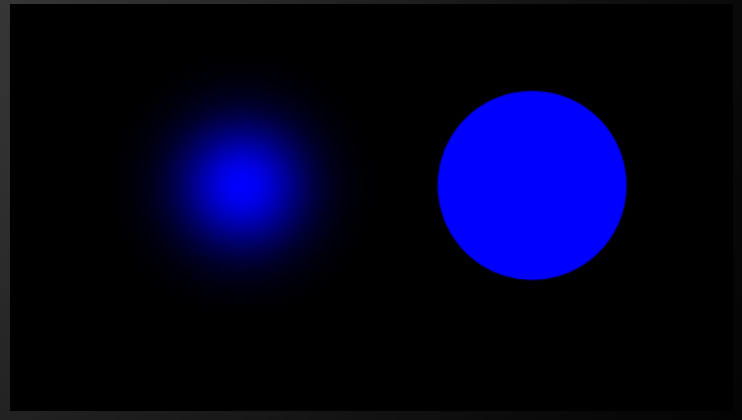


green.bmp ↑

red.bmp ↓



blue.bmp ↓

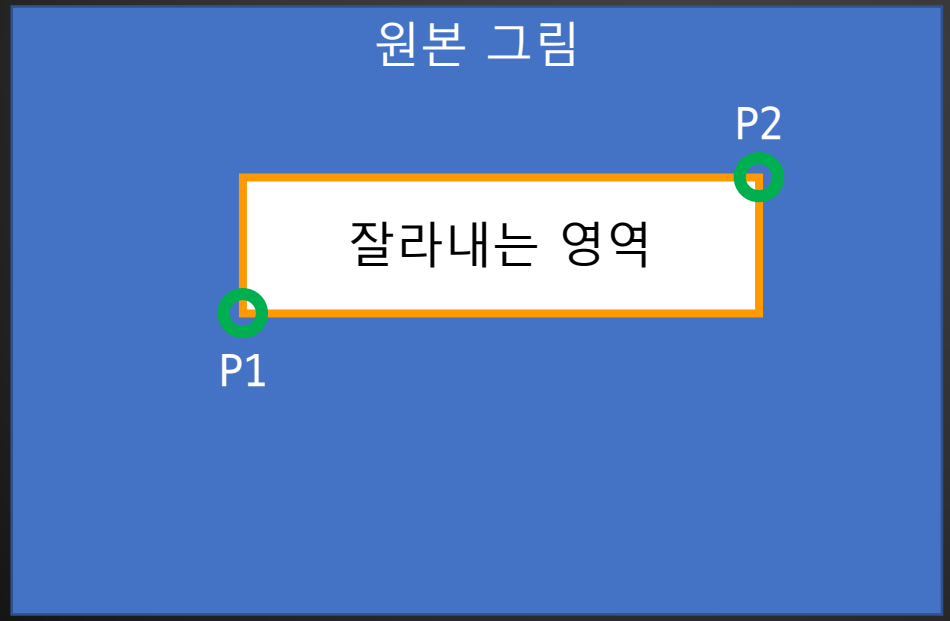


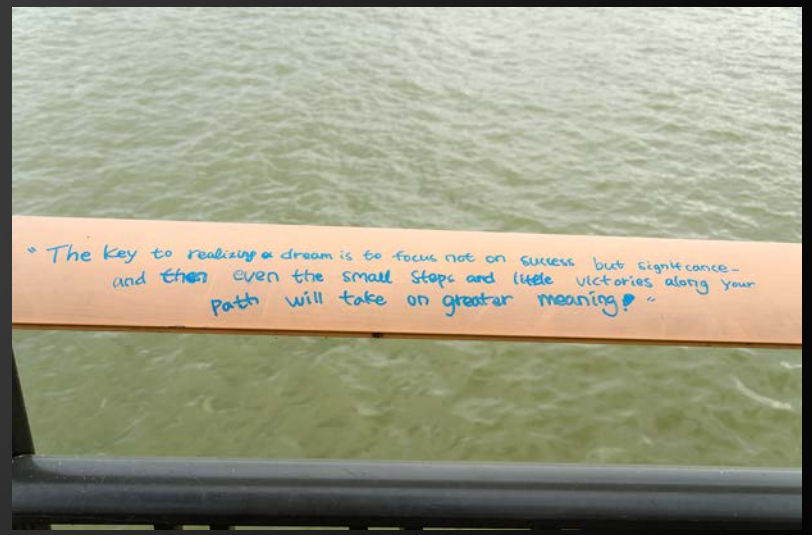
• 예제 3 - crop()

➢함께 배포한 ori.bmp파일에서 일부를 잘라내어 새로운 그림을 만듭니다.

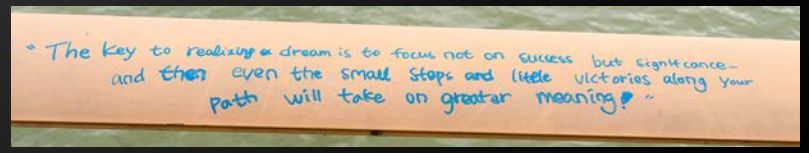
➢두 개의 점을 입력 받아서, 두 점으로 직사각형을 만들고, 만든 직사각형을 기준으로 그림을 잘라서 새로운 파일을 만듦.

```
RUN crop()

File size : 3015992
offset : 54
Image Size : (1232X816)
24 BPP
P1 x : 0
P1 y : 282
P2 x : 1231
P2 y : 500
P1 :    0,  282
P2 : 1231,  500
Check crop.bmp!!!
```





ori.bmp ↑



crop.bmp ↑

• 예제 4 - paletteswap()

➢ 함께 배포한 logo.bmp 파일의 팔레트 값을 사용자가 입력한 값으로 바꿔 다른 색을 보여줍니다.

```
RUN paletteswap()

File size : 82900
offset : 74
Image Size : (949X174)
4 BPP
Palette has 5 color
Original palette 0 B : 255
Original palette 0 G : 255
Original palette 0 R : 255

Original palette 1 B : 255
Original palette 1 G : 255
Original palette 1 R : 0

Original palette 2 B : 255
Original palette 2 G : 0
Original palette 2 R : 255

Original palette 3 B : 0
Original palette 3 G : 255
Original palette 3 R : 255

Original palette 4 B : 0
Original palette 4 G : 0
Original palette 4 R : 0
```

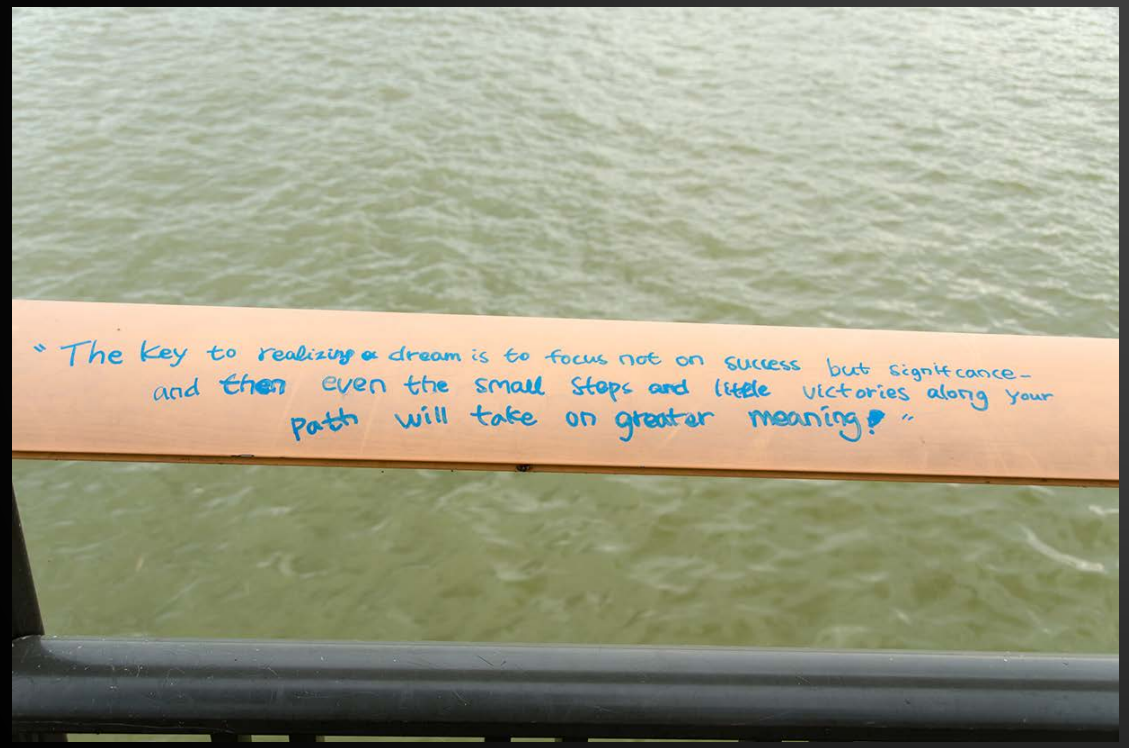
```
input palette 0 B : 0
input palette 0 G : 0
input palette 0 R : 0

input palette 1 B : 255
input palette 1 G : 0
input palette 1 R : 0

input palette 2 B : 0
input palette 2 G : 255
input palette 2 R : 0

input palette 3 B : 0
input palette 3 G : 0
input palette 3 R : 255

input palette 4 B : 0
input palette 4 G : 0
input palette 4 R : 0

Check logo_swap.bmp!!
```

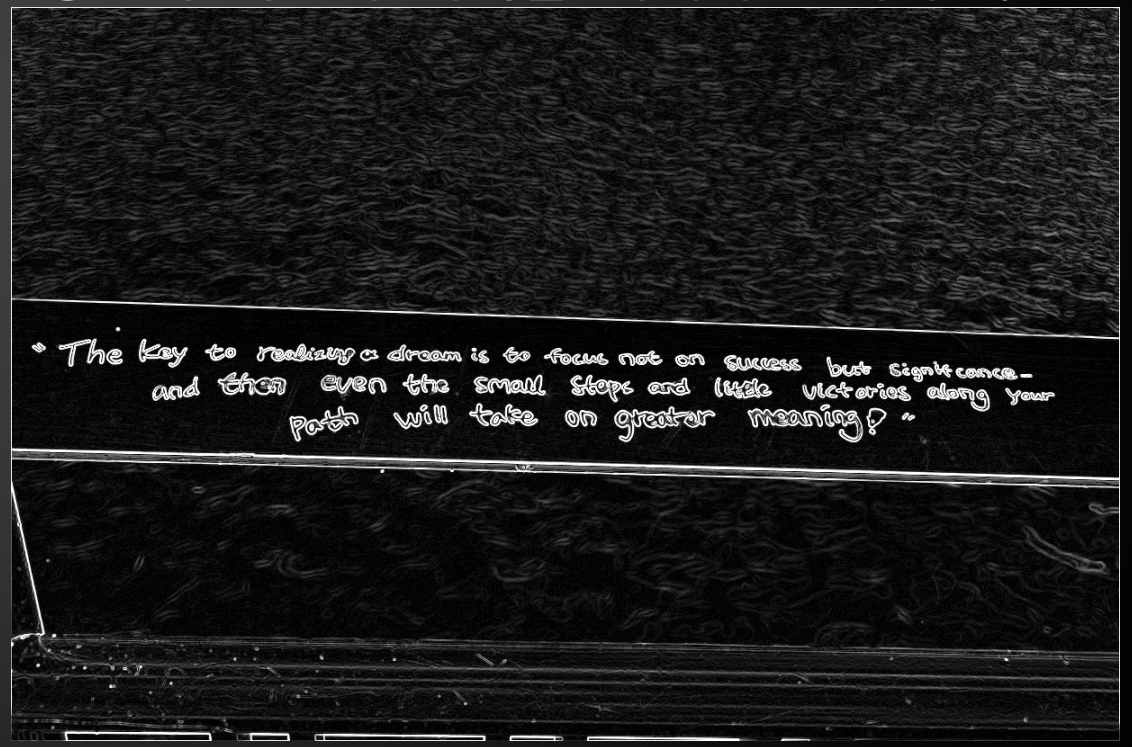
logo.bmp ↑

logo_swap.bmp ↓

•과제

- ori.bmp에 대하여 EdgeDetection을 진행합니다.(egdt함수 작성)
  - 참고 : Sobel Edge
  - 구현과정에서 overflow 조심하세요!!
    - (2^8 = 256, 연산 값이 255를 초과하는 경우에 대하여 대책을 세워야 합니다!!)





•과제 부연 설명

➢Sobel Edge 연산

➢아래와 같은 행렬을 준비한다.

- ➢각각 X방향, Y방향 edge를 계산할 때 사용한다.
- ➢각각 X방향 커널, Y방향 커널이라고 한다.

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

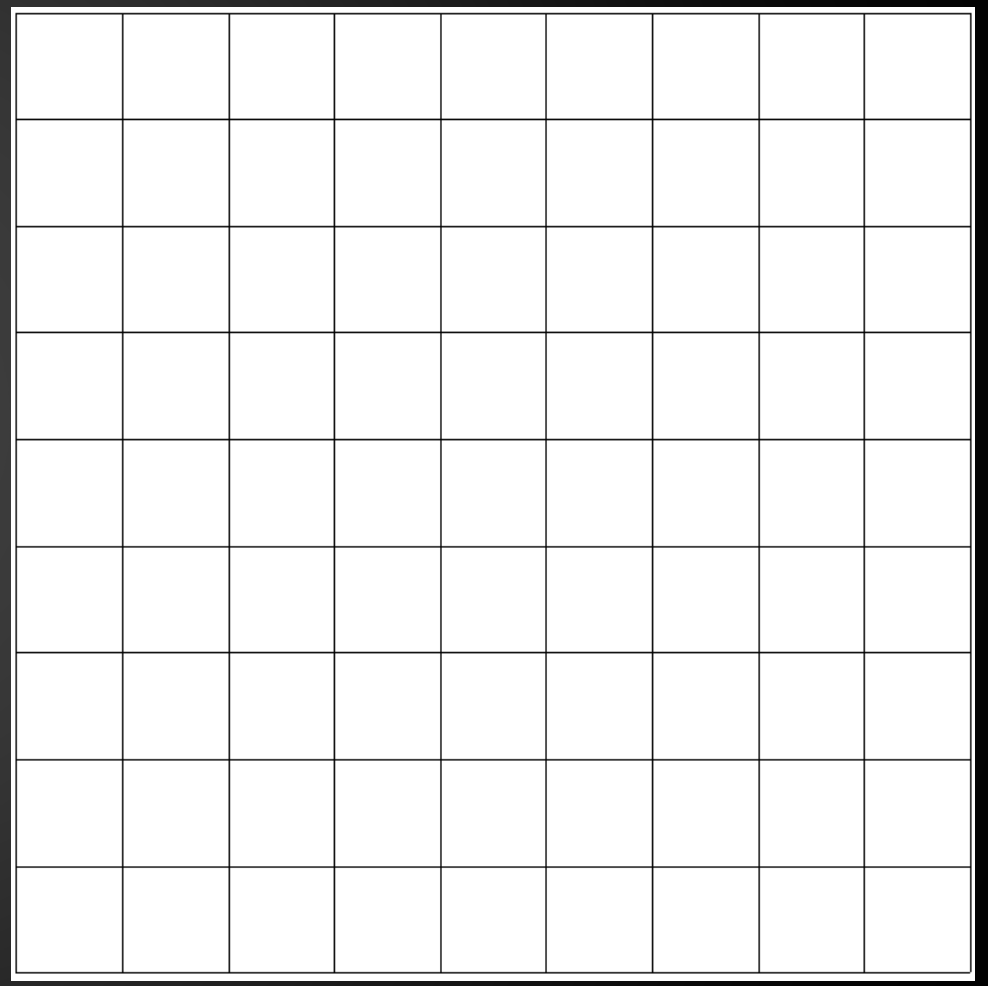
•과제 부연 설명

- Sobel Edge 연산
  - 원본 그림이 오른쪽과 같이 있다고 가정한다.
  - 하얀 배경에 검은 선이 가로방향으로 그려져 있다.
  - 각 칸 안의 숫자는 픽셀 값이다.
  - 본 Sobel Edge 연산은 흑백 영상에 대하여 진행한다고 가정한다.
    - 과제도 흑백으로 진행해도 됩니다.

255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0
255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255

•과제 부연 설명

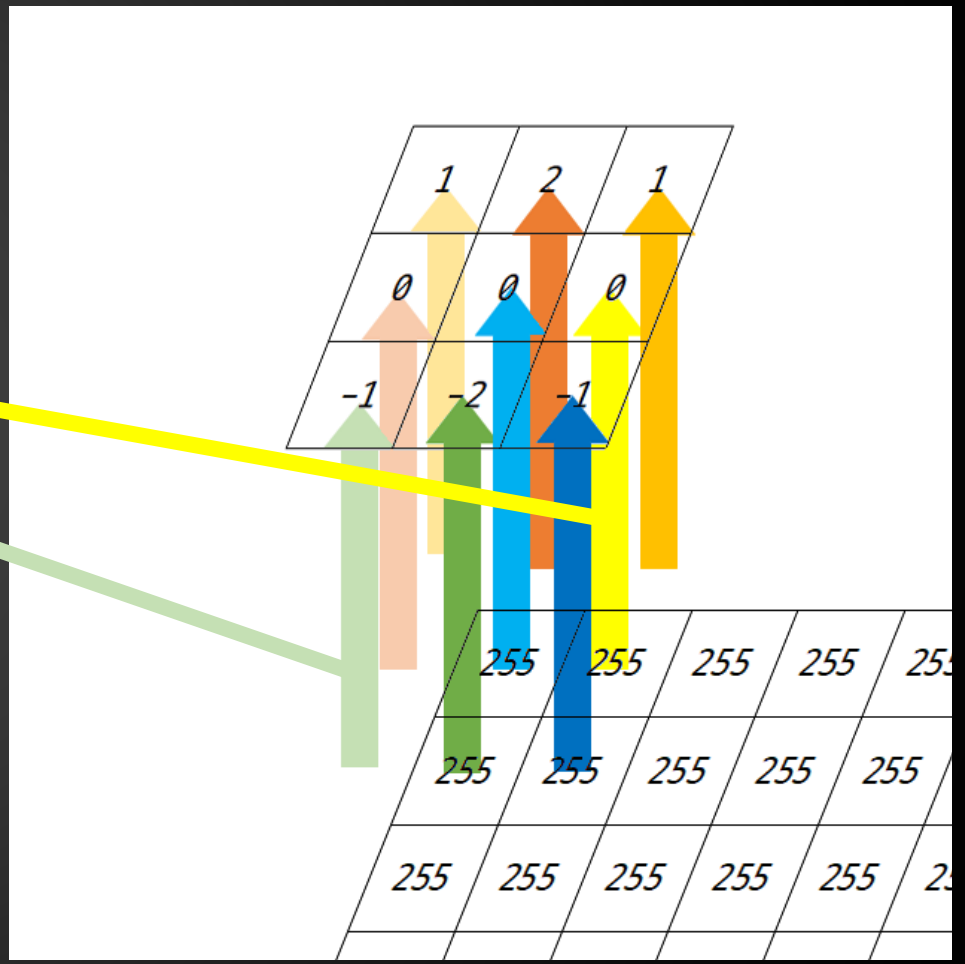
- Sobel Edge 연산
  - 원본 그림과 같은 픽셀 개수의 메모리 영역을 두개 준비한다
    - 메모리의 크기는 다음 슬라이드를 보고 생각한다.
      - (* int, *char...)
    - 반드시 이차원 배열형태일 필요는 없다. 일차원배열처럼 사용하되, 인덱스 값을 잘 바꾸면서 사용해도 된다.
      - 이번 강의의 모든 예제들은 일차원배열을 사용하였다.
    - 각 메모리 영역은 X 방향, Y 방향의 Sobel Edge 강도를 저장한다.
  - 왜 배열을 만들지 않고, 메모리 동적할당을 하나요?
    - 입력 영상이 얼마나 큰지 모르니까!



• 과제 부연 설명

➢ Sobel Edge 연산

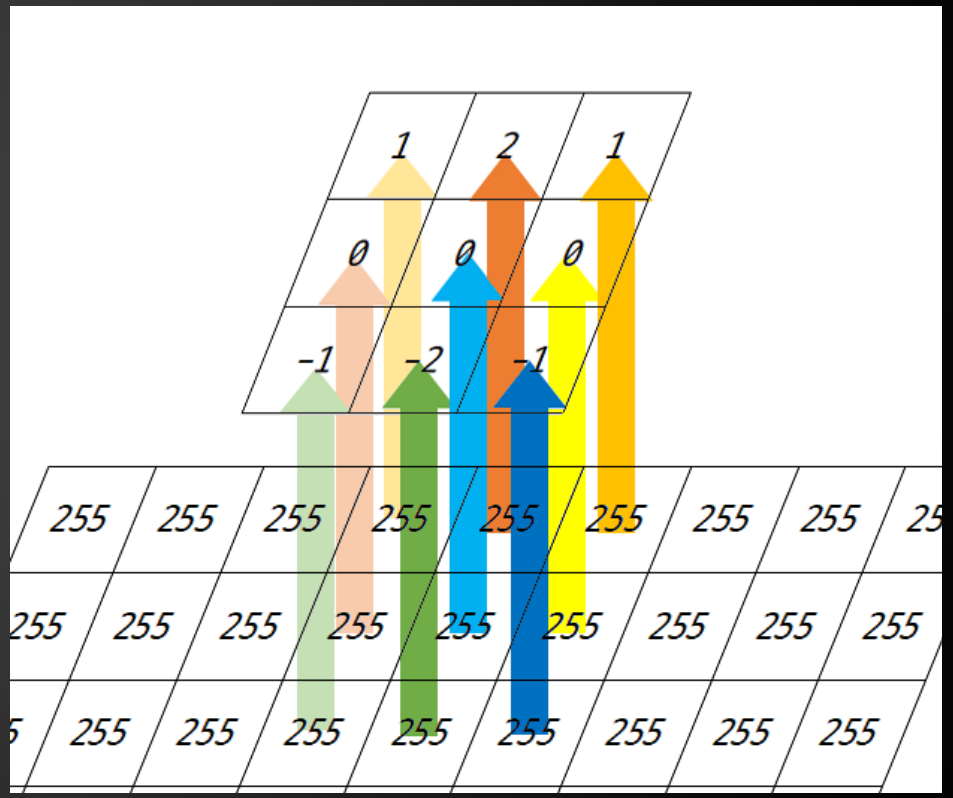
- ➢ 커널을 움직이면서 원본 그림과 correlation 계산을 진행한다.
  - ➢ 커널 값 * 원본 그림의 값을 계산한다.
    - ➢ 우선 Y축 방향 커널을 사용한다고 생각하자.
    - ➢ 커널이 3X3이므로 총 9번의 곱셈 연산을 한다.
    - ➢ 예를 들어 노란 화살표에서는 255 * 0 = 0이 발생한다.
    - ➢ 원본 값이 없는 경우, 그 원본 값은 0으로 가정한다.
  - ➢ 계산한 값을 모두 더한다.
  - ➢ 더한 값을 앞에서 만든 메모리 영역에 저장한다.
  - ➢ 우측 그림을 참고한다.
    - ➢ 우측 그림은 더한 결과 값이 -765이다.
    - ➢ 메모리 영역 선언할 때 어떤 자료형으로 선언해야 할까요?(*int, *float, *char...)



•과제 부연 설명

➢Sobel Edge 연산

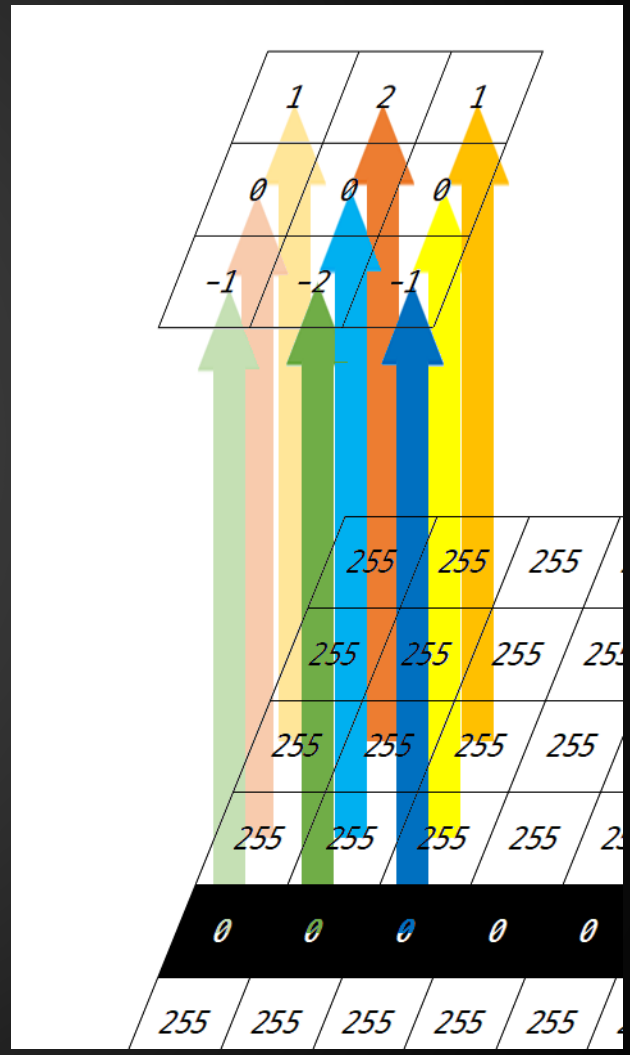
- ➢커널을 움직이면서 원본 그림과 correlation 계산을 진행한다.
  - ➢커널을 움직이면서 연산을 반복한다.
    - ➢우측 그림은 연산 결과 값이 0이다.
    - ➢메모리에 잘 저장하자.



•과제 부연 설명

➢Sobel Edge 연산

- ➢커널을 움직이면서 원본 그림과 correlation 계산을 진행한다.
  - ➢커널을 움직이면서 연산을 반복한다.
    - ➢우측 그림은 연산 결과 값이 1020이다.
      - ➢edge가 발생하는 부분이므로 연산 값이 크다.
    - ➢메모리에 잘 저장하자.



•과제 부연 설명

- Sobel Edge 연산
  - 커널을 움직이면서 원본 그림과 correlation 계산을 진행한다.
    - 최종 연산 결과는 오른쪽과 같다.
    - 검은 선이 지나가는 부분의 바로 위아래에 큰 값이 존재함을 알 수 있다.
      - Edge의 존재 확인
    - X축 방향 커널을 사용하여 연산을 한번 더 진행하여 다른 배열에 그 값을 저장하자.

-765	-1020	-1020	-1020	-1020	-1020	-1020	-1020	-765
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
765	1020	1020	1020	1020	1020	1020	1020	765
0	0	0	0	0	0	0	0	0
-765	-1020	-1020	-1020	-1020	-1020	-1020	-1020	-765
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
765	1020	1020	1020	1020	1020	1020	1020	765

•과제 부연 설명

-765	0	0	0	0	0	0	0	765
-1020	0	0	0	0	0	0	0	1020
-1020	0	0	0	0	0	0	0	1020
-765	0	0	0	0	0	0	0	765
-510	0	0	0	0	0	0	0	510
-765	0	0	0	0	0	0	0	765
-1020	0	0	0	0	0	0	0	1020
-1020	0	0	0	0	0	0	0	1020
-765	0	0	0	0	0	0	0	765

X축 방향 커널을 사용하여 연산한 결과

-765	-1020	-1020	-1020	-1020	-1020	-1020	-1020	-765
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
765	1020	1020	1020	1020	1020	1020	1020	765
0	0	0	0	0	0	0	0	0
-765	-1020	-1020	-1020	-1020	-1020	-1020	-1020	-765
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
765	1020	1020	1020	1020	1020	1020	1020	765

Y축 방향 커널을 사용하여 연산한 결과

•과제 부연 설명

- X축 방향 edge 강도와 Y축 방향 edge 강도를 사용하여 벡터 합을 계산한다.
  - 서로 대응되는 픽셀의 강도끼리 벡터 합을 계산한다.
  - 예를 들어 0,0의 edge강도는 root((-765)^2+(-765)^2) = 1081.87이다.
  - 예를 들어 0,4의 edge강도는 root((-510)^2+0^2) = 510이다.
    - (좌측 상단을 0,0으로 가정한다.)
  - 계산한 값을 다시 0~255범위로 재계산한다.
    - 최대 edge 강도를 찾아서 그 값을 k라 하면, 전체 edge 강도에 대해 255/k를 곱해 준다.
  - 재계산한 값을 이용하여 비트맵 파일을 생성한다.

•과제 부연 설명

1081.873	1020	1020	1020	1020	1020	1020	1020	1081.873
1020	0	0	0	0	0	0	0	1020
1020	0	0	0	0	0	0	0	1020
1275	1020	1020	1020	1020	1020	1020	1020	1275
1020	0	0	0	0	0	0	0	1020
1275	1020	1020	1020	1020	1020	1020	1020	1275
1020	0	0	0	0	0	0	0	1020
1020	0	0	0	0	0	0	0	1020
1081.873	1020	1020	1020	1020	1020	1020	1020	1081.873

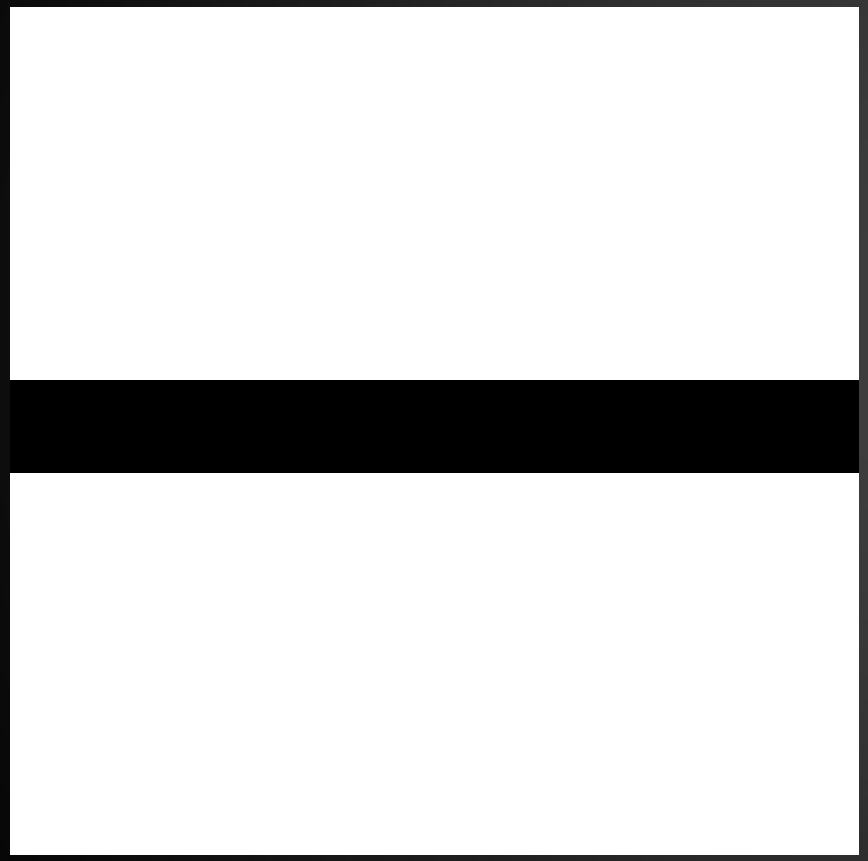
백터 합 연산 결과(최대값 : 1275)

216.3747	204	204	204	204	204	204	204	216.3747
204	0	0	0	0	0	0	0	204
204	0	0	0	0	0	0	0	204
255	204	204	204	204	204	204	204	255
204	0	0	0	0	0	0	0	204
255	204	204	204	204	204	204	204	255
204	0	0	0	0	0	0	0	204
204	0	0	0	0	0	0	0	204
216.3747	204	204	204	204	204	204	204	216.3747

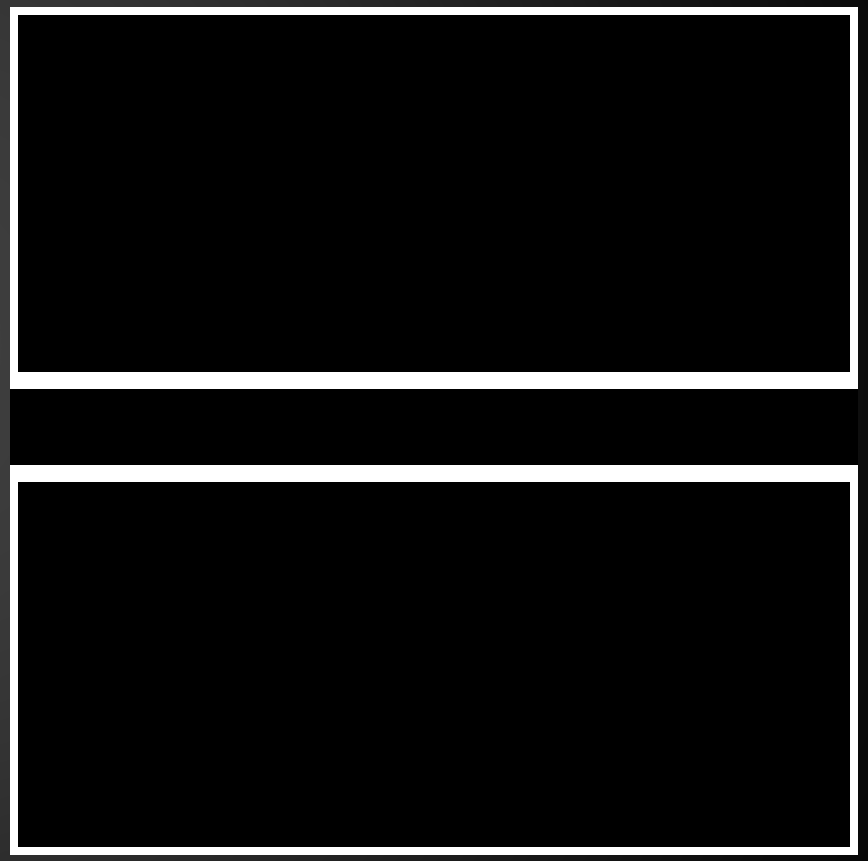
0 ~ 255로 재할당한 결과

•과제 부연 설명

➢시각화



원본



edge 연산 결과