**BMP(\*.bmp) Manual**

Codesys 팀

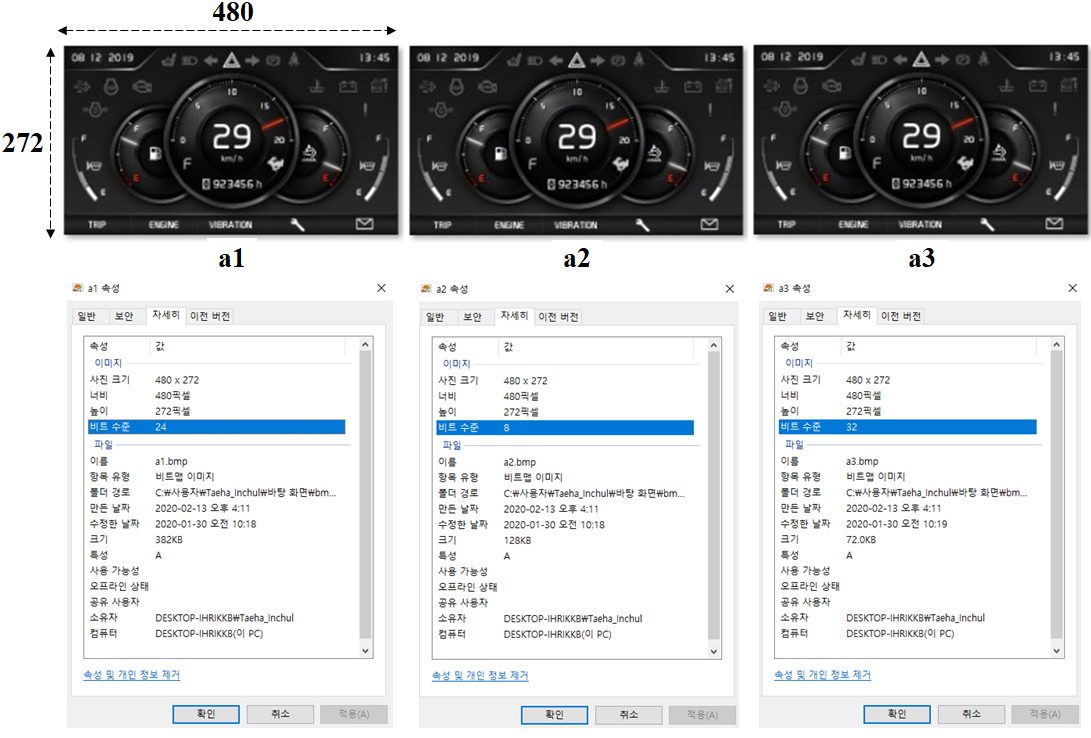
백인철

1. **BMP 파일(\*.bmp)**

이미지 파일의 확장자는 여러가지가 있지만, BMP(\*.bmp)에 대해서 설명한다.

BMP 파일은 마이크로소프트 윈도우 운영체제에서 쓰는 그림 파일의 확장자이다. 마이크로소프트(MS)가 개발한 그래픽 파일 포맷으로, 그래픽 파일 형식 중 가장 단순한 구조를 가지고 있다. ‘비트맵’이라고 읽고 BMP, DIB 등의 확장자를 사용한다.

다음 그림을 확인하면 세 가지의 파일(a1, a2, a3)이 있다. 세 파일을 확인해보면 같은 이미지 인 것을 확인 할 수 있다. 이미지는 가로 세로 인데 각 파일의 크기(용량)가 다르다.



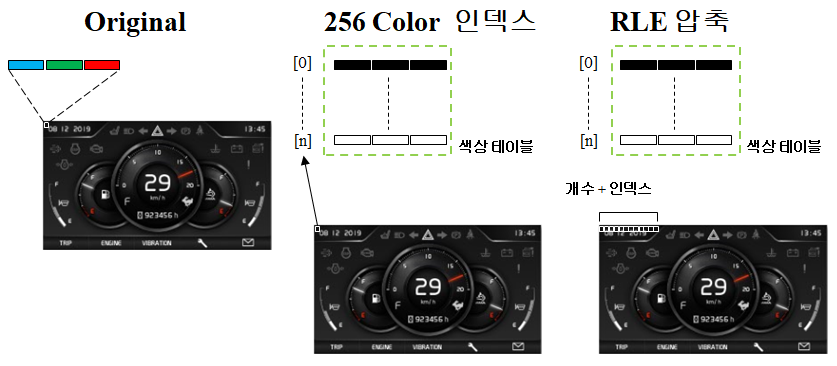
육안으로 확인했을 때 같은 이미지이지만 크기(용량)의 차이가 나는 이유는 각 비트맵 파일의 구조이다. 따라서 각 파일의 구조가 어떻게 이루어졌는지, 그리고 어떻게 색을 나타내는지에 대해서 파악해야 한다.

위 그림의 비트맵 a1은 원본 비트맵 파일이다. 비트맵 a2는 원본 비트맵 이미지를 256 Color 인덱스를 사용하여 변환한 파일이고, 마지막 비트맵 a3는 RLE 압축을 한 파일이다.

원본 비트맵 이미지는 한 개의 픽셀에 색을 표시하기 위해 B, G, R 각각 8 Bit 총 24 Bit(3 Byte)로 구성된다. 즉 한 픽셀에 색은 파랑 – 초록 – 빨강 순으로 각각 1 Byte로 표현할 수 있다.

256 Color 인덱스를 사용하여 변환한 비트맵 이미지 a2는 원본 비트맵 이미지 a1에서 사용된 색 데이터를 24 Bit 기준 크기 순으로 나열하고 인덱스화 한다. 이렇게 인덱스 화 시킨 색 데이터를 ‘색상 테이블’이라 부른다. 각 픽셀에 색을 출력하기 위해서 색상 테이블의 인덱스 값을 데이터로 사용한다.

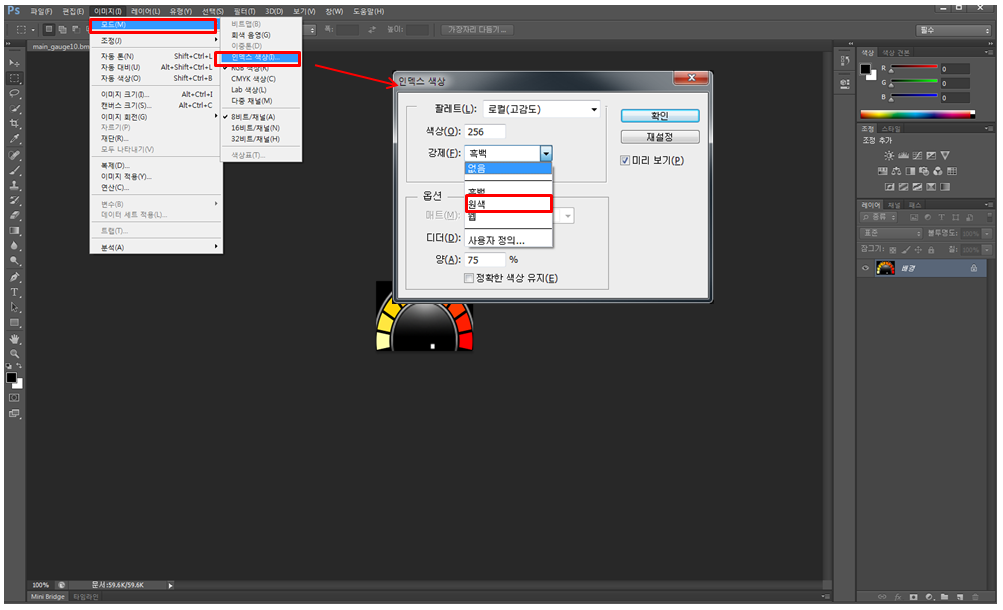
RLE로 압축한 비트맵 이미지 a3는 비트맵 이미지 a2와 동일하게 색상 테이블을 사용한다. 그리고 색을 나열하다 보면 같은 색을 반복적으로 나열하는 경우가 있다. 예시로 든 이미지 또한 검은색이 반복적으로 나오는 부분이 있는데 반복이 되는 색을 그대로 나열하는 것이 아니라 반복되는 색의 개수를 명시해준다.

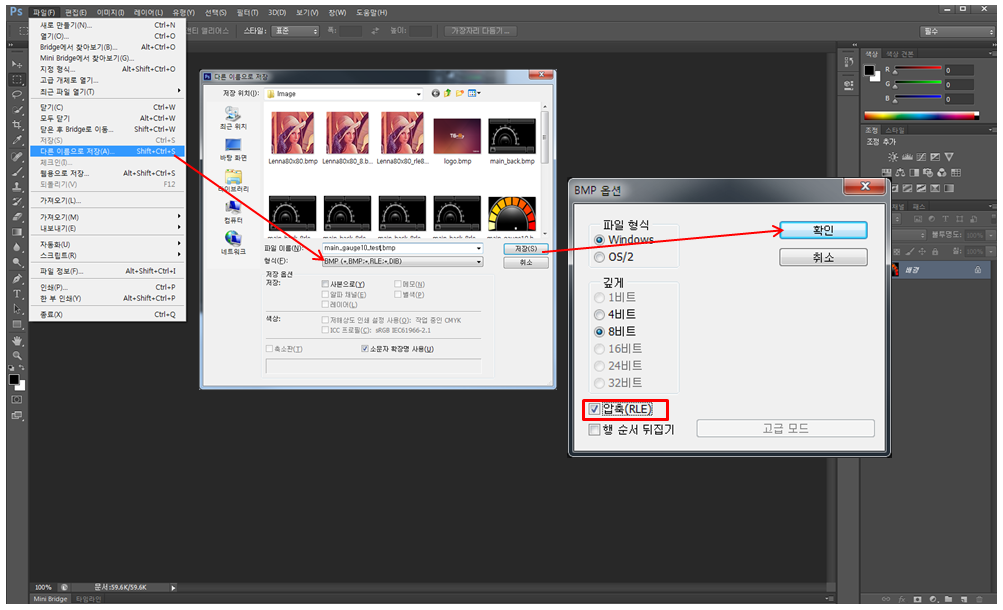


비트맵 이미지를 LCD에 이미지를 출력할 때 같은 그림이지만 원본 이미지를 출력할 때 보다 RLE 압축된 이미지를 출력하는 것이 메모리를 절약할 수 있다.

1. **포토샵을 이용한 RLE 압축**

포토샵을 이용하면 비트맵 이미지를 RLE 압축 할 수 있다. RLE 압축 방법은 다음과 같다.





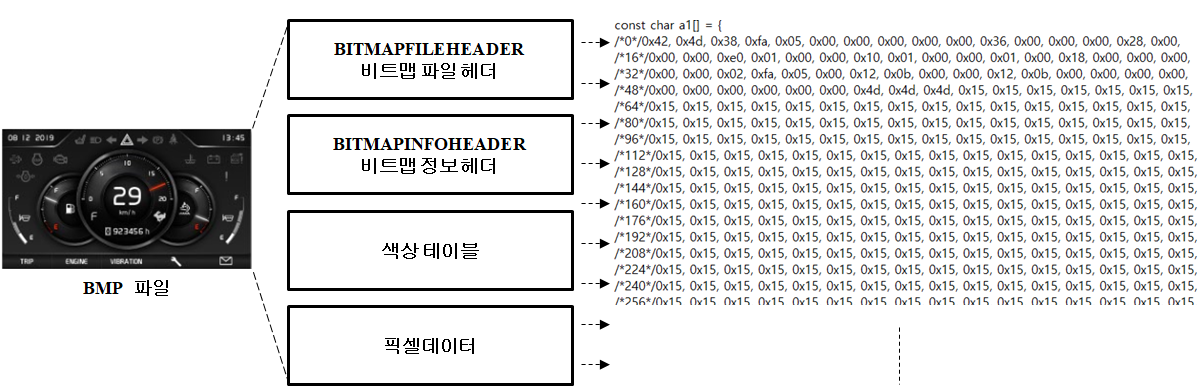
1. **RLE 압축 및 Turbo C++(BMP2H.C)**

비트맵 파일의 내부는 바이너리 형식으로 메모장과 같은 텍스트 편집기로 열어서 확인이 불가능하다. 바이너리 형식으로 확인할 수 있는 프로그램을 인터넷에서 다운받아 내부 바이너리 형식의 내용을 확인할 수 있다.

또한 BMP 이미지 그대로 C 코드에 첨부할 수 없다. C 코드에서 BMP 이미지를 확인하기 위해서는 BMP 파일(\*.bmp)를 C 코드(\*.H) 헤더파일로 변환을 해야 한다. 첨부한 BMP2H.C를 Turbo C++에서 실행하여 BMP 이미지 파일을 변환한다.

1. **BMP 파일 구조**

비트맵 파일의 바이너리 구조는 간단하게 다음과 같다.



**비트맵 파일 헤더(BITMAPFILEHEADER)**는 BMP 파일에 대한 일반 정보를 담고 있다. **비트맵 정보 헤더(BITMAPINFOHEADER)**는 비트맵 그림에 대한 정보를 가지고 있다. **색상 테이블**은 인덱스 컬러 비트맵에 쓰이는 색의 정의를 담고 있다. 따라서 비트맵 헤더에 색상 테이블이 존재하지 않을 수 있다. **픽셀데이터**는 그림의 픽셀 단위로 실제 그림을 담고 있다. 픽셀데이터의 형태는 압축의 형태에 따라 달라진다.

비트맵 파일 헤더와 비트맵 정보 헤더는 다음과 같이 규격이 정해져 있다.

(예시)

**const char a1[] = {**

**/\*0\*/0x42, 0x4d, 0x38, 0xfa, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x36, 0x00, 0x00, 0x00, 0x28, 0x00,**

**/\*16\*/0x00, 0x00, 0xe0, 0x01, 0x00, 0x00, 0x10, 0x01, 0x00, 0x00, 0x01, 0x00, 0x18, 0x00, 0x00, 0x00,**

**/\*32\*/0x00, 0x00, 0x02, 0xfa, 0x05, 0x00, 0x12, 0x0b, 0x00, 0x00, 0x12, 0x0b, 0x00, 0x00, 0x00, 0x00,**

**/\*48\*/0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4d, 0x4d, 0x4d, 0x15, 0x15, 0x15, 0x15, 0x15, 0x15, 0x15, …**

* **비트맵 파일 헤더(BITMAPFILEHEADER)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| **매직 넘버** | | **BMP 파일 크기 [Byte]** | | | | **Reserved1** | | **Reserved2** | | **데이터 시작 오프셋** | | | |

비트맵 파일 헤더는 총 14 Byte이다.

매직 넘버란 BMP 파일의 식별자로 0x42, 0x4d는 ASCII 코드로 해석하면 B, M이다.

데이터 시작 오프셋이란 비트맵 데이터(픽셀 데이터)의 시작 오프셋으로 픽셀 데이터가 시작하는 곳의 위치를 파악 할 수 있다.

* **비트맵 정보 헤더(BITMAPINFOHEADER)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** |
| **정보 헤더 크기(40 Byte)** | | | | **비트맵 가로 [Pixel]** | | | | **비트맵 세로 [Pixel]** | | | | **색판 수(1)** | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **28** | **29** | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** |
| **비트 수** | | **압축 방식** | | | | **그림 크기** | | | | **그림의 가로 해상도** | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **42** | **43** | **44** | **45** | **46** | **47** | **48** | **49** | **50** | **51** | **52** | **53** |
| **그림의 새로 해상도** | | | | **색 팔레트의 색 수** | | | | **중요한 색의 수** | | | |

비트맵 정보 헤더는 총 40 Byte이다.

압축 방식은 다음 표를 확인한다.

|  |  |  |
| --- | --- | --- |
| 값 | 식별자 | 압축 방식 |
| 0 | BI\_RGB | 없음 |
| 1 | BI\_RLE8 | RLE 8비트 |
| 2 | BI\_RLE4 | RLE 4비트 |
| 3 | BI\_BITFIELDS | 비트 필드 |
| 4 | BI\_JPEG | JPEG |
| 5 | BI\_PNG | PNG |

\*\* 위 포토샵을 이용하여 RLE 압축을 하면 압축 방식은 1(BI\_RLE8)이 된다.

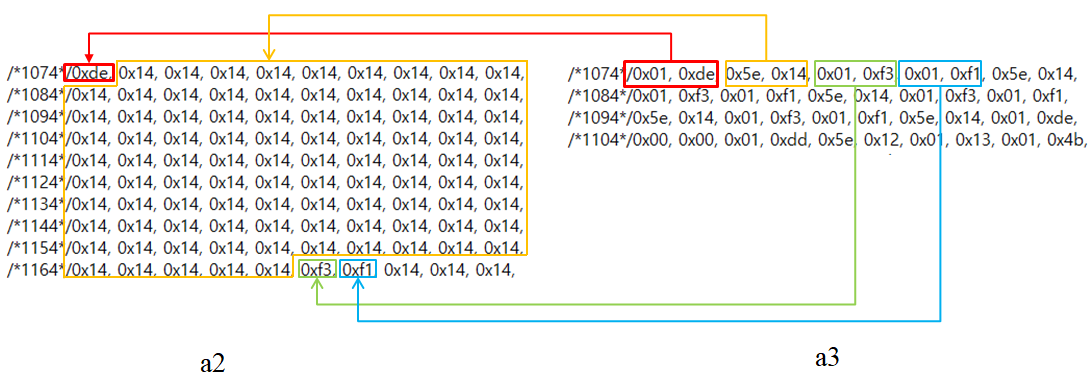
첨부한 비트맵 이미지 a1, a2, a3를 C 코드로 변환하고 내부 헤더의 값을 확인해보면 이해하기 쉽다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | A1 | A2 | A3 |
| 매직 넘버 | 0x42, 0x4d | 0x42, 0x4d | 0x42, 0x4d |
| BM | BM | BM |
| BMP 파일 크기 | 0x38, 0xfa, 0x05, 0x00 | 0x34, 0x02, 0x02, 0x00 | 0x2c, 0x20, 0x01, 0x00 |
| 391736 [Byte] | 161636 [Byte] | 73772 [Byte] |
| Reserved1 | 0x00, 0x00 | 0x00, 0x00 | 0x00, 0x00 |
| Reserved2 | 0 | 0 | 0 |
| 데이터 시작 오프셋 | 0x36, 0x00, 0x00,0x00 | 0x32, 0x04, 0x00, 0x00 | 0x32, 0x04, 0x00, 0x00 |
| 54 | 1074 | 1074 |
| 정보헤더크기 | 0x28, 0x00, 0x00, 0x00 | 0x28, 0x00, 0x00, 0x00 | 0x28, 0x00, 0x00, 0x00 |
| 40 [Byte] | 40 [Byte] | 40 [Byte] |
| 비트맵 가로 | 0xe0, 0x01, 0x00, 0x00 | 0xe0, 0x01, 0x00, 0x00 | 0xe0, 0x01, 0x00, 0x00 |
| 480 | 480 | 480 |
| 비트맵 세로 | 0x10, 0x01, 0x00, 0x00 | 0x10, 0x01, 0x00, 0x00 | 0x10, 0x01, 0x00, 0x00 |
| 272 | 272 | 272 |
| 색판 수 | 0x01, 0x00 | 0x01, 0x00 | 0x01, 0x00 |
| 1 | 1 | 1 |
| 비트 수 | 0x18, 0x00 | 0x08, 0x00 | 0x08, 0x00 |
| 24 [Bit] | 8 [Bit] | 8 [Bit] |
| 압축 방식 | 0x00, 0x00, 0x00, 0x00, | 0x00, 0x00, 0x00, 0x00, | 0x01, 0x00, 0x00, 0x00, |
| 0(BI\_RGB) | 0(BI\_RGB) | 1(BI\_RLE8) |
| 그림 크기 | 0x02, 0xfa, 0x05, 0x00 | 0x02, 0xfe, 0x01, 0x00 | 0xfa, 0x1b, 0x01, 0x00 |
| 391,682 [Byte] | 130,562 [Byte] | 72,698 [Byte] |
| 그림의 가로 해상도 | 0x12, 0x0b, 0x00, 0x00 | 0x12, 0x0b, 0x00, 0x00 | 0x12, 0x0b, 0x00, 0x00 |
| 2834 | 2834 | 2834 |
| 그림의 세로 해상도 | 0x12, 0x0b, 0x00, 0x00 | 0x12, 0x0b, 0x00, 0x00 | 0x12, 0x0b, 0x00, 0x00 |
| 2834 | 2834 | 2834 |
| 색 팔레트의 색 수 | 0x00, 0x00, 0x00, 0x00 | 0xff, 0x00, 0x00, 0x00 | 0xff, 0x00, 0x00, 0x00 |
| 0 | 255 | 255 |
| 중요한 색의 수 | 0x00, 0x00, 0x00, 0x00 | 0xff, 0x00, 0x00, 0x00 | 0xff, 0x00, 0x00, 0x00 |
| 0 | 255 | 255 |

데이터 시작 오프셋 정보를 통해 색 데이터가 어디에서부터 시작하는지를 알 수 있다. 예를 들어 비트맵 이미지 a1은 54번째부터, 비트맵 이미지 a2는 1074번째부터, 비트맵 이미지 a3는 1074번째이다. 비트맵 이미지는 원본 이미지이므로 비트맵 이미지 헤더가 끝나는 시점부터 바로 색 데이터가 나열되지만, 비트맵 이미지 a2, a3는 비트맵 이미지 헤더 뒤에 색상 테이블이 존재한다. 색상 테이블이 끝나는 시점부터 색 데이터가 나열된다. 비트맵 이미지 a2, a3의 색상 테이블을 비교하면 같은 이미지이므로 두 색상 테이블이 같음을 확인할 수 있다. 또한 54번째부터 시작하는 색상 테이블은 4번째마다 0x00이 존재하는데 이는 비트맵 픽셀의 색을 지정하기 위해 3 Byte를 사용하고 4 번째에 0x00을 위치시켜 모든 요소가 짝수 주소에 위치될 수 있도록 하였다. 그리고 이 색상 테이블의 색 순서는 Blue 🡪 Green🡪 Red🡪 Reserved(0x00)이다. Blue, Green, Red, Reserved를 하나의 픽셀데이터(색 데이터)로 인식하고 인덱스 화 하였다.

1. **픽셀데이터(색 데이터) 비교!**

각 비트맵 이미지의 헤더를 파악하고, 픽셀 데이터(색 데이터)가 시작하는 위치를 파악했으므로 어떻게 압축된 이미지가 실제 이미지와 다르지 않은지 확인해본다. 순서 비교는 비트맵 이미지 a3에서 a2, a1 순으로 비교한다.



비트맵 이미지 a3의 픽셀 데이터 처음 부분은 0x01, 0xde이다. 첫 번째 0x01은 뒤에 나오는 데이터의 개수를 의미한다. 즉, 0xde가 0x01개 이다. 비트맵 이미지 a2와 비교하면 0xde가 한 개 있다. 이때 0xde는 실제 색이 아니라 인덱스 값이다. 즉 실제 색은 색상 테이블의 0xde번째에 있다는 것이다. 색상 테이블은 Blue, Green, Red, Reserved 구성되고 이 4 개가 하나의 인덱스를 이루기 때문에 위치는 다음과 같이 찾을 수 있다.

54 Byte로 구성된 비트맵 이미지 헤더 뒤에 색상 테이블이 나오기 때문에 54를 더해준다. 이 규칙으로 색을 찾으면, 가 된다. 942번부터 4개의 데이터를 확인해보면 0x4d, 0x4d, 0x4d, 0x00이다. 이를 비교하기 위해 비트맵 이미지 a1의 데이터 시작 부분에서 확인 해보면 0xd4, 0x4d, 0x4d인 것을 확인할 수 있다. 이와 같이 BI\_RLE8 압축 방식에서는 최소한 2 Byte로 데이터를 표현한다. 이러한 규칙을 가지고 값을 비교하다 보면 규칙이 적용되지 않는 부분이 존재하는데, 이는 특수 코드 명령어가 있기 때문에다. BI\_RLE8 압축 방식에서 사용되는 특수 코드 명령어는 다음과 같다.

첫 번째 바이트가 0(0x00)이면 다음에 나오는 바이트가 명령어이다. 즉 0x00은 특수 코드(escape code)이다.

|  |  |
| --- | --- |
| 0x00, 0x00 | 현재 스캔 라인의 마지막을 표시하고, 새로운 스캔 라인을 시작한다. |
| 0x00, 0x01 | 비트맵 픽셀을 구성하는 데이터의 마지막을 표시하고, 픽셀의 압축 상태를 더 이상 해제시키지 않는다. |
| 0x00, 0xnn, 0xk1,..., 0xkn | 절대 모드. 0xnn이 3<= 0xnn <=255 라는 범위에 있으면, 0xk1에서부터 0xkn까지 압축되지 않은 픽셀을 0xnn개 출력한다. 전체 저장 단위는 짝수 개의 바이트를 포함해야 하므로, 만약 0xnn이 홀수라면 마지막에 0x00이 추가된다. |

첫 번째 바이트가 항상 0으로 지정되어 있는 특수 코드는 다른 방법으로 처리되어야 한다. 저장 단위의 두 번째 바이트는 명령어이거나 또는 0에서부터 255개까지의 서로 다른 값이다 가장 간단한 특수 코드의 하나는 현재 스캔 라인의 마지막을 표시하는 것이다. 프로그램은 두 개의 0x00을 읽어 들일 때 다음 픽셀을 새로운 스캔 라인에서부터 시작하여 출력할 것이다. 0x01과 같은 두 번째 바이트는 비트맵 화상을 구성하는 압축된 데이터의 마지막을 표시한다. 그리하여, 프로그램은 더 이상 데이터의 압축 상태를 해제시킬 필요가 없다. 마지막인 특수 코드 명령어는 압축된 연속적인 픽셀을 절대 모드(absolute mode)로 저장한다. 이것은 압축되더라도 저장 영역을 절약하는 것이 힘들거나 불가능한 서로 다른 색상의 인접한 몇 개의 픽셀을 표현하는 경우에 유용하다. 절대 모드에서 첫 번째 바이트는 0x00이고, 두 번째 바이트는 3에서부터 255까지의 범위를 가지는 n이다. 그 다음에는 압축되지 않는 정확히 n개의 픽셀이 존재한다. 만약 n이 홀수라면, 다음 바이트가 짝수 주소에서 시작할 수 있도록 마지막에 0x00이 추가된다. 다음은 절대 모드일 때, 예시이다.

0x00, 0x04, 0x03, 0x0a, 0x01, 0x02

첫 번째 바이트는 일반적인 특수 코드를 뜻하는 0x00이다. 두 번째 바이트는 절대 모드를 뜻하는 것으로, 3보다 크거나 같은 값이다. 이 두 번째 바이트는 얼마나 많은 연속적인 픽셀이 나타날 것인지 지정하는 것으로, 여기에서는 네 바이트가 연속되어 있음을 알 수 있다. 네 바이트의 각각은 압축되지 않는 상태로 저장된 픽셀의 색상 값의 인덱스이다. 프로그램은 이 저장 단위의 압축 상태를 해제시켜서 색상 값인 0x03, 0x0a, 0x01, 0x02를 사용하는 네 개의 픽셀을 출력한다.

절대 모드에서 픽셀의 개수가 홀수이면, 아래의 예제에서처럼 마지막에 추가되는 0을 무시할 수 있다.

0x00, 0x03, 0x04, 0x02, 0x0e, 0x00

0x00, 0x07, 0x08, 0x09, 0x09, 0x01, 0x04, 0x09, 0x02, 0x00

0x00, 0x05, 0x0b, 0x01, 0x04, 0x03, 0x08, 0x00

첫 번째 줄은 인덱스 값 0x04, 0x02, 0x0e를 가지는 세 개의 픽셀을 뜻한다. 명령어의 길이를 짝수 바이트로 만들기 위해서 추가된 마지막의 0x00은 무시 할 수 있다. 두 번째 줄은 7 개의 픽셀을 저장하고 있다. 세 번째 줄은 5 개의 픽셀을 저장하고 있다. 각각의 줄은 무시해도 되는 0x00으로 끝난다. 그러나, 절대 모드의 마지막 바이트가 0x00의 값을 사용하여 픽셀의 색상 값 인덱스를 표현하는 것이 가능하다. 또한, 절대 모드에서 픽셀의 인덱스를 표현하기 위해 0x00이라는 값이 언제든지 사용될 수 있을 것이다. 예를 들어, n의 값이 짝수라면, n 다음의 모든 바이트는 의미 있는 값이다.

0x00, 0x04, 0x05, 0x00, 0x03, 0x00  
이 명령어는 인덱스 값인 0x05, 0x00, 0x03, 0x00을 가지는 네 개의 픽셀을 표현한다. 여기서 마지막에 있는 0은 무시되지 않아야 한다.

이러한 특수 코드까지 고려하여 코드를 작성해야 한다.

1. **비트맵 이미지 출력하기!**

출력하고자 하는 비트맵 이미지에 대하여 C 코드 헤더파일을 다음과 같이 include한다.



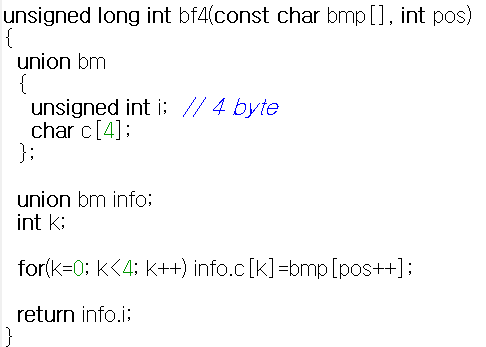
비트맵 이미지를 출력하기 위해 아래와 같이 함수를 선언한다.



함수 실행 시, LCD에 출력하는 비트맵 이미지에 대한 기본 정보를 얻기 위해 다음과 같이 한다.



네 바이트로 이루어진 비트맵 이미지 헤더 정보를 계산하기 위해서 bf4() 함수를 다음과 같이 선언하였다.



Bf4() 함수는 네 바이트로 이루어진 바이너리(헥사) 값을 계산하는 함수이다. 이 함수를 통해서 픽셀 데이터의 시작 위치와 비트맵 이미지의 가로, 세로 길이를 확인한다. 함수의 입력인자로는 두 개를 사용한다. 첫 번째 인자로는 배열을 사용한다. 비트맵 이미지는 C코드에서 배열로 이루어져 있기 때문에 원하는 비트맵 이미지를 선택하기 위해서 배열을 입력인자로 사용한다. 두 번째 인자로 비트맵 이미지의 헤더 구성 중, 비트맵 픽셀 데이터의 시작위치, 비트맵 가로, 세로 길이를 알 수 있는 오프셋 값을 입력 인자로 받기 위해서 정수형 변수를 인자로 설정하였다.



While(1) 루프 문을 이용해 바이트 단위 픽셀 데이터를 선택해야 한다.

변수 number와 number2를 이용하여 픽셀 데이터를 선택한다. 이때 number의 값이 0x00(특수 코드)일 때와 0x00이 아닐 때로 구분하여 생각한다.

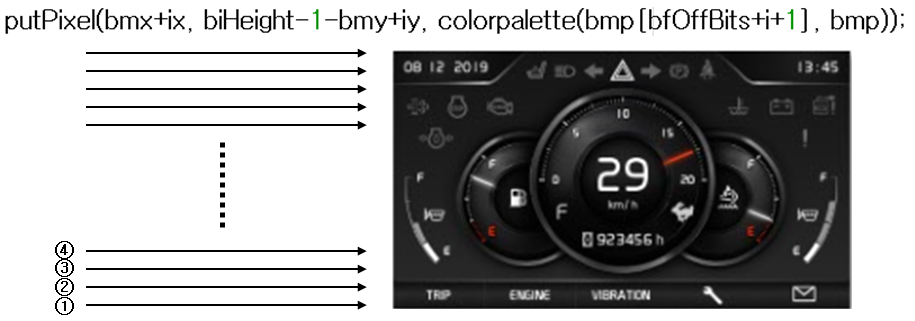
먼저 number의 값이 0x00으로 특수 코드 인 경우, 그 다음 픽셀 데이터를 number2에 저장한다. number2가 0x00이면 스캔 라인의 마지막을 표시하므로 아무런 동작을 하지 않고 인덱스 값(i)을 2 증가시킨다. 인덱스 값을 2 증가 시키는 이유는 0x00, 0x00을 건너 뛰어야 하기 때문이다. Number2가 0x01이면 픽셀 데이터의 마지막을 의미한다. 따라서 무한 루프로 동작하는 while(1) 루프 문을 종료시키는 역할을 수행(break;)하도록 한다. number2가 3에서 255사이인 경우 절대 모드이다.

절대 모드일 경우, number2는 for문의 횟수를 지정하는 변수로 작용한다. 절대 모드는 횟수 데이터 다음부터 number2의 수만큼 연속으로 색상 테이블의 인덱스 값을 나타낸다. 따라서 따라서 기존 인덱스 값(i)에 특수 코드와 횟수 데이터의 수(2)를 더하고 for문의 인덱스를 더해야 순차적으로 데이터를 선택할 수 있다. 픽셀에 색 데이터를 지정하기 위해서 putPixel함수를 사용하였다. colorpalette()함수를 사용하여 색상 테이블의 인덱스 값을 계산하고, B-G-R 3 Byte를 R-G-B 2 Byte로 바꿔준다. colorpalette()함수는 다음과 같다. R-G-B 2 Byte로 바꾸기 위해서 R-G-B에 대하여 각각 비트 이동을 한다.

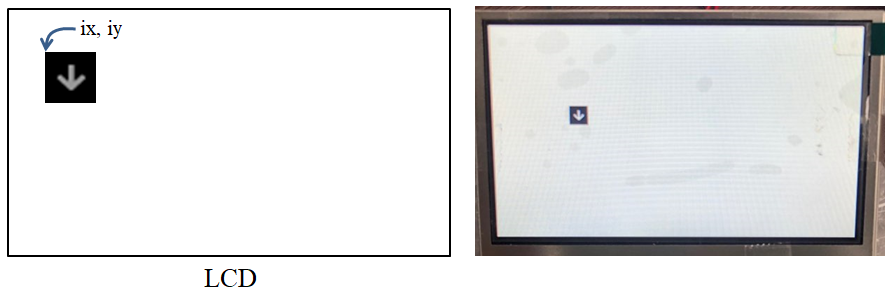


number2가 짝수 인 경우, 특수 코드(0x00), 횟수 데이터, Number2만큼 인덱스 값을 증가시켜주면 된다. 하지만 홀수 인 경우, 데이터의 시작을 짝수로 맞추기 위해 마지막에 추가되는 0x00을 건너뛰어주어야 한다. 따라서 인덱스 값에 특수 코드(0x00), 횟수 데이터, Number2의 수 그리고 1(0x00)을 더해주어야 한다.

특수 코드가 아닌 경우는 number2에 픽셀 데이터 값을 저장한다. 그리고 number의 값만큼 반복해서 픽셀 데이터를 출력한다.



LCD에 출력할 때, putPixel() 입력 인자 중 첫 번째는 LCD의 x축 위치, 두 번째는 LCD의 y축 위치이다. biHeight는 LCD의 세로 축 길이로 0부터 271까지 총 272이다. 비트맵 이미지의 x축은 LCD의 x축과 방향이 일치하지만 y축은 LCD의 y축과 반대이다. y축은 거꾸로 출력해야 한다. y축만 거꾸로 출력하기 위해서 biHeight-1에서 bmy를 빼준다.



BMP\_draw() 함수의 두 번째, 세 번째 입력인자는 그림의 위치 값으로 LCD 화면 내에서 이미지를 원하는 위치에 그릴 수 있다.

지금까지 RLE 압축된 BMP Image LCD출력 설명을 마칩니다.

Codesys팀 백인철