영상 파일 형식(1)

- RAW
  - 영상의 픽셀 값만을 저장하는 형식
  - 영상을 읽기 위해서는 먼저 영상의 정보(해상도 및 색상의 수)를 알고 있어야 한다.
- BMP(BitMaP)
  - 비트맵 디지털 그림을 저장하는 데 쓰이는 그림 파일 포맷
    - BMP는 1비트 흑백부터 24비트까지의 색 농도를 지원
  - Windows 비트맵 파일 형식은 다른 Microsoft Windows 프로그램과 호환
  - 파일 압축을 지원하지 않아서 파일 용량 크다 → 웹 페이지에 적합하지 않다.
    - 사진 이미지의 경우 PNG 파일, JPEG 파일이 좋다.

영상 파일 형식(2)

- GIF(Graphics Interchange Format)
  - GIF 애니메이션으로 많이 사용되는 8비트(256색)의 인터넷 표준 파일
    - 투명성, 압축, 인터레이스 및 복수 이미지 그림(애니메이션 GIF) 지원
    - 알파 채널 투명성에서 지원하는 것 같은 반투명 효과나 페이드 효과를 지원하지 않는다.
  - 무손실 압축이고 256색의 색상표만 지원 → 최근 트루 칼라 지원
    - 색 정보가 손실되어 JPEG 형식이나 PNG 형식보다 품질이 떨어진다.
- TIFF(Tagged Image File Format)
  - 인쇄 업계에서 가장 널리 지원되는 그래픽 파일 형식
    - Macintosh 컴퓨터와 Windows 기반 컴퓨터 사이에서 많이 사용
  - 무손실 압축과 태그를 지원하는 최초의 이미지 포맷
    - 선택적 압축을 지원, 무손실 압축 포맷 사용 가능 → 의료 쪽에서 사용
    - 확장 가능한 형식이므로 프로그래머가 기능을 추가하거나 특정 요구 사항을 충족하기 위해 기본 사양을 수정 가능

영상 파일 형식(3)

- JPEG(Joint Photographic Experts Group)
  - 비트맵 이미지 파일, 이미지 손실 압축
    - 24비트 색으로 최고 수준의 압축을 제공하는 플랫폼 독립적 형식의 손실 압축 방식 → 광범위하게 지원되는 인터넷 표준
    - (압축 과정에서 손실 발생) 인터레이스(Progressive JPEG 파일)가 지원
  - 가변 압축을 통해 파일 크기를 줄일 수 있지만, JPEG 파일 크기를 줄이면 이미지 품질이 저하 (압축 비율은 최대 100:1까지 가능)
    - JPEG 형식은 10:1에서 20:1의 비율로 그림 품질을 거의 저하시키지 않고 파일을 안정적으로 압축
      - 압축 비율이 5:1이면서 그림 무결성이 크게 손실되는 경우도 있다.
  - 손실이 있는 압축이기 때문에 원본 그림 데이터의 품질이 저하
    - 색이 적거나 비슷한 색으로 된 넓은 영역이 있거나 밝기가 크게 대비되는 단순한 그림에는 적합하지 않다.
  - JPEG2000 : JPEG보다 이미지 압축률, 화질 등이 개선된 압축 / 무손실 압축도 지원

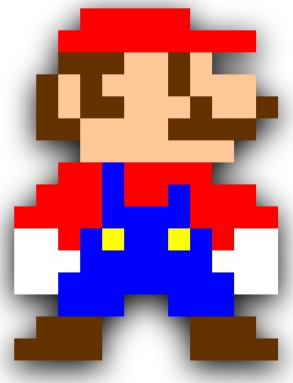
영상 파일 형식(4)

- PNG(Portable Network Graphics)
  - 플랫폼 독립적 형식으로 웹 브라우저에서도 지원
    - 상위 수준의 무손실 압축, 알파 채널 투명성, 감마 조정, 인터레이스 지원
    - GIF에 특허권 문제가 발생하자, GIF 보다 더 좋고, 새로운 그림 파일 형식을 개발한 것 / 무손실 압축 지원
  - JPEG(트루 칼러)와 GIF(투명 배경과 움짤)의 장점을 제공
- EPS(Encapsulated PostScript)
  - 전문적인 고품질 이미지 인쇄에 필요한 벡터 파일 포맷
    - 인쇄용으로 설계 된 업계 표준 형식이며, 캡슐화된 포스트스크립트 형식은 벡터 정보와 비트맵 정보를 모두 나타낼 수 있는 독점적인 프린터 설명 언어
- OpenEXR : ILM에서 개발한 HDR 이미지 포맷
- WepP : 구글에서 만든 이미지 파일 포맷
  - 웹 고속화를 위해 개발된 압축 포맷

영상 파일 형식(5)

- PBM, PGM, PPM : PBM(이진), PGM(흑백), PPM(컬러)
  - PPM(Portable PixMap) : PGM, PBM과는 달리 컬러영상을 표현할 수 있는 파일
  - PGM(Portable Gray Map Image) : 픽셀 당 1 또는 2 바이트 (8 또는 16 비트)로 인코딩 된 그레이 스케일 이미지 파일
  - PBM(Portable BitMap) : 흑백 이미지 파일

```
import DataPixels from "./DataPixels.js";

const C = "255, 0, 0";      //Hat & Shirt
const B = "100, 50, 0";     //Brown Hair & Boots
const S = "255, 200, 150";  //Skin Tone
const O = "0, 0, 255";      //Blue Overalls
const Y = "255, 255, 0";    //Yellow Buckles
const W = "255, 255, 255";  //White Gloves
const _ = "0, 0, 0, 0";     //Transparent (RGBA Format)

const data = [[_, _, _, C, C, C, C, C, _, _, _, _],
              [_, _, C, C, C, C, C, C, C, C, _, _],
              [_, _, B, B, B, S, S, B, S, _, _, _],
              [_, B, S, B, S, S, S, B, S, S, S, _],
              [_, B, S, B, B, S, S, S, B, S, S, B],
              [_, B, B, S, S, S, S, B, B, B, _, _],
              [_, _, _, S, S, S, S, S, S, S, _, _],
              [_, _, C, C, O, C, C, C, C, _, _, _],
              [_, C, C, C, O, C, C, O, C, C, _, _],
              [C, C, C, C, O, O, O, O, C, C, C, C],
              [W, W, C, O, Y, O, O, Y, O, C, W, W],
              [W, W, W, O, O, O, O, O, O, W, W, W],
              [W, W, O, O, O, O, O, O, O, O, W, W],
              [_, _, O, O, O, _, _, O, O, O, _, _],
              [_, B, B, B, _, _, _, _, B, B, B, _],
              [B, B, B, B, _, _, _, _, B, B, B, B]];

const size = 30;

const mario = new DataPixels(data, size).image;
mario.style.filter = "drop-shadow(0 10px 20px #000000)";

document.body.appendChild(mario);
```



PBM/PGM/PPM 이미지

- PBM, PGM, PPM 영상의 구성
  - 헤더와 데이터부로 구성
  - 헤더는 ASCII 코드로 되어있어 일반 텍스트 편집기로 볼 수 있다.
  - PBM(Portable BitMap) : 흑백 영상
  - PGM(Portable GrayMap) : 그레이 영상
  - PPM(Portalble PixMap) : 컬러 영상
    - PPM 형식의 경우 한 픽셀 당 R, G, B 의 세 값이 저장
- 이미지의 포맷
  - 매직넘버 : 파일유형이 무엇이며 데이터가 어떻게 저장되어 있는가를 정의
  - 영상의 너비(Width)
  - 영상의 높이(Height)
  - 최대값 : 최대 명암도 등급 또는 컬러 채널 값(PBM 파일은 최대값 항목 없음)

PBM/PGM/PPM 형식

- 헤더
  - 파일의 유형(매직넘버값)
  - 영상의 너비 영상의 높이
  - 최대 명암도/컬러 값
  - 각각의 값은 white space(space, tab, line feed, carriage return)로 구분

```
예)
    P5
    640  480
    255
```

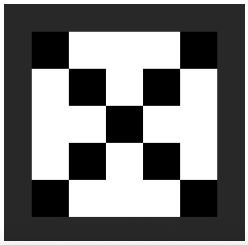
- 파일의 유형
  - 매직넘버값은 파일의 데이터가 ASCII 문자이거나 2진수인가에 따라 다르다.

형식	ASCII 문자	이진수
PBM	P1	P4
PGM	P2	P5
PPM	P3	P6

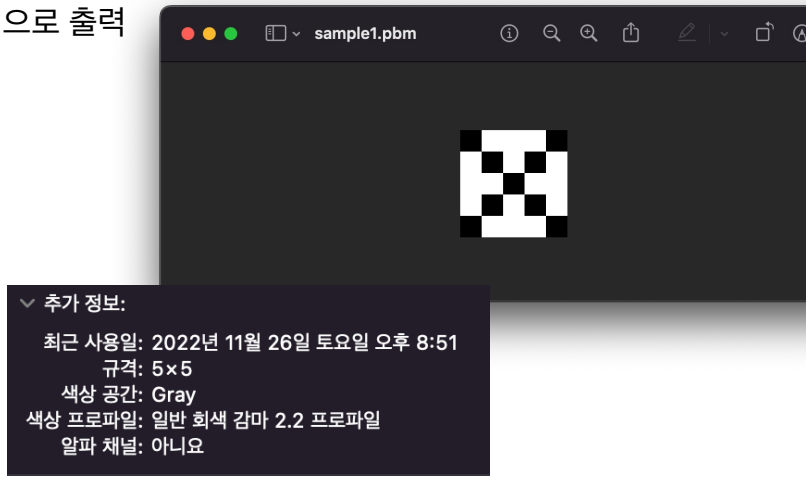
```
P1 5 5
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
```

PBM 파일 생성

- PBM(Portable BitMap) : 흑백영상
  - 5 x 5 크기의 매우 작은 이미지
  - vi 등의 텍스트 에디터를 이용해서 작업
  - sample1.pbm
    - 1 : 검은색, 0 : 흰색으로 출력



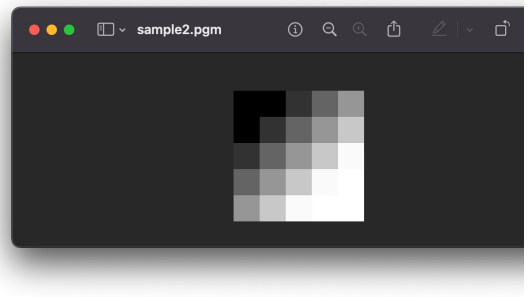
```
P1
5 5
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
```

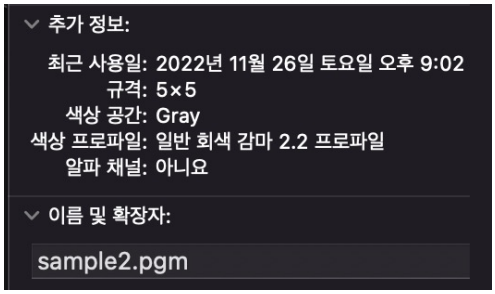


PGM 파일 생성

- PGM(Portable GrayMap) : 그레이 영상
  - 5 x 5 크기의 매우 작은 이미지
  - sample2.pgm
    - 1 : 검은색, 255 : 흰색, 중간 값 : 희색

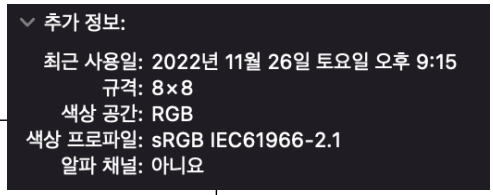
```
P2
5 5
255
  0   0  50 100 150
  0  50 100 150 200
 50 100 150 200 250
100 150 200 250 255
150 200 250 255 255
```



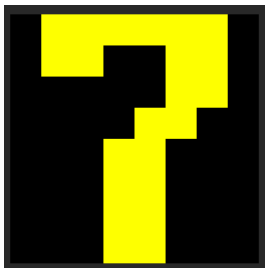


PPM 파일 생성

- PPM(Portalble PixMap) : 컬러 영상
  - 8 x 8 크기의 매우 작은 이미지

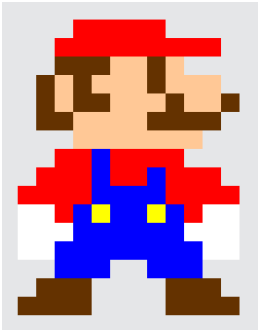



```
P3
8 8
1
0 0 0 110 110 110 110 110 110 0 0 0
0 0 0 110 110 0 0 0 0 0 0 110 110 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 110 110 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 110 110 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 110 110 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 110 110 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 110 110 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 110 110 0 0 0 0 0 0 0 0 0
```

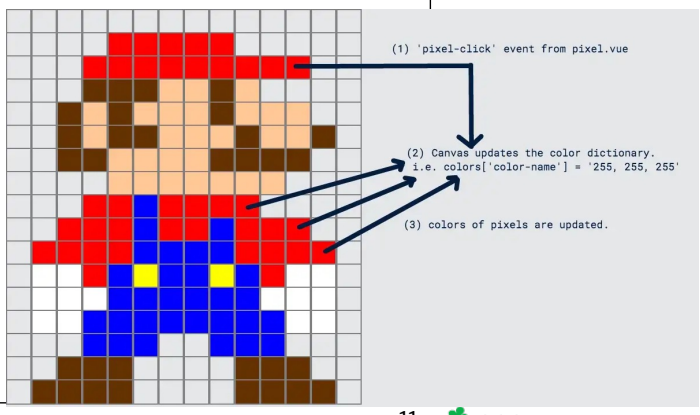


미니 프로젝트(1)

- 마리오 파일을 PPM 파일로 작성해보시오.



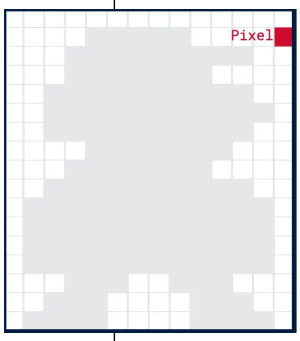
```
const C = "C";          // Hat & Shirt
const B = "B";          // Brown Hair & Boots
const S = "S";          // Skin Tone
const O = "O";          // Blue Overalls
const Y = "Y";          // Yellow Buckles
const W = "W";          // White Gloves
const _ = "_";
this.colors = {
   [C]: "255, 0, 0",
   [B]: "100, 50, 0",
   [S]: "255, 200, 150",
   [O]: "0, 0, 255",
   [Y]: "255, 255, 0",
   [W]: "255, 255, 255",
   [_]: "229, 230, 232"};
```



서영진 valentis@chollian.net

미니 프로젝트(2)

```
this.pixelData = [
   [_, _, _, _, _, _, _, _, _, _, _, _, _, _, _, _],
   [_, _, _, _, C, C, C, C, C, _, _, _, _, _, _, _],
   [_, _, _, C, C, C, C, C, C, C, C, C, _, _, _, _],
   [_, _, _, B, B, B, S, S, B, S, _, _, _, _, _, _],
   [_, _, B, S, B, S, S, S, B, S, S, S, _, _, _, _],
   [_, _, B, S, B, B, S, S, S, B, S, S, B, _, _, _],
   [_, _, B, B, S, S, S, S, B, B, B, B, _, _, _, _],
   [_, _, _, _, S, S, S, S, S, S, S, _, _, _, _, _],
   [_, _, _, C, C, O, C, C, C, C, _, _, _, _, _, _],
   [_, _, C, C, C, O, C, C, O, C, C, C, _, _, _, _],
   [_, C, C, C, C, O, O, O, O, C, C, C, C, _, _, _],
   [_, W, W, C, O, Y, O, O, Y, O, C, W, W, _, _, _],
   [_, W, W, W, O, O, O, O, O, O, W, W, W, _, _, _],
   [_, W, W, O, O, O, O, O, O, O, O, W, W, _, _, _],
   [_, _, _, O, O, O, _, _, O, O, O, _, _, _, _, _],
   [_, _, B, B, B, _, _, _, _, B, B, B, _, _, _, _],
   [_, B, B, B, B, _, _, _, _, B, B, B, B, _]];
```



서영진 valentis@chollian.net

PBM 파일 읽기(1)

- PBM의 헤더를 분석한 후 데이터를 로드해서 표시
  - PBM 파일과 PGM 파일을 읽는 법은 큰 차이가 없다.
    → PGM 파일은 헤더에 명암도의 최대값만 더 포함
- 파일 포맷을 위한 구조체

파일의 유형(매직넘버값)
영상의 너비 영상의 높이
최대 명암도/컬러 값

```
#include <stdio.h>
#include <stdlib.h>

typedef unsigned char uchar;

typedef struct {
   char MN[2];                  /* 파일의 유형(매직넘버값) */
   int  width, height;          /* 영상의 너비   영상의 높이 */
   uchar **pixelData;           /* 이미지 데이터의 값 */
} PBMImage;

enum BoolData { FALSE, TRUE };
```


PBM 파일 읽기(2)

- 2개의 함수를 선언
  - readPBM( ) 함수 : PBM 이미지 파일명과 구조체 변수
  - releasePBM( ) 함수 : readPBM( ) 함수에서 동적으로 할당한 메모리 해제

```
int readPBM(char* filename, PBMImage* img);
void releasePBM(PBMImage* img);

int main(int argc, char** argv)       /* 명령행 인수로 파일명을 받아온다. */
{
   PBMImage img;
   int x, y;

   if(argc != 2) {                    /* 파일명이 없으면 프로그램 종료 */
      fprintf(stderr, "usage : %s <filename>\n", argv[0]);
      return -1;
   }
```

PBM 파일 읽기(3)

- readPBM( ) 함수에서 파일을 읽어서 정보를 읽어오고 읽어온 데이터를 터미널로 출력 후 메모리 정리

```
    /* 이미지 파일 읽기 */
    if(readPBM(argv[1], &img) != TRUE) {
        return -1;
    }

    /* 이미지를 터미널로 출력 */
    for(x = 0; x < img.height; x++) {
        for(y = 0; y < img.width; y++)
            printf("%s", (img.pixelData[x][y] == 1)?"□":"■");
        printf("\n");
    }

    releasePBM(&img);
    return 0;
}
```


PBM 파일 읽기(4)

- readPBM( ) 함수
  - 파일을 읽어서 PBM 파일을 분석
    1. 파일 열기
    2. 매직넘버 읽기
    3. 가로(width), 세로(height) 읽기
    4. 가로 x 세로만큼 메모리 할당
    5. 파일로부터 픽셀값 읽어오기
    6. 파일 닫기

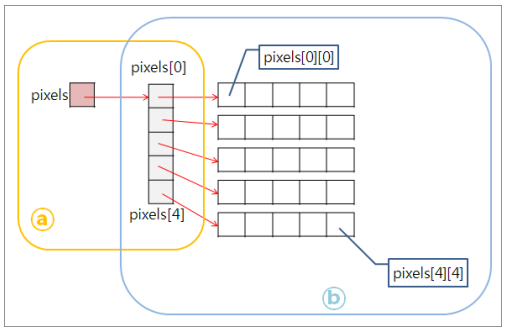
```
int readPBM(char* filename, PBMImage* img)
{
    FILE* fp;
    unsigned int x, y, tmp;                    /* 한 픽셀은 unsigned char로 표현 */

    fp = fopen(filename, "r");                 /* 파일 열기 */
    if(fp == NULL) {
```

PBM 파일 읽기(5)

```
    fprintf(stderr, "파일을 열 수 없습니다 : %s\n", filename);
    return FALSE;
}

fscanf(fp, "%c %c", &img->MN[0], &img->MN[1]);  /* 매직넘버 읽기 */
if(img->MN[0] != 'P' || img->MN[1] != '1') {
    fprintf(stderr, "PBM 이미지 포맷이 아닙니다 : %s\n", img->MN);
    return FALSE;
}

fscanf(fp, "%d %d", &img->width, &img->height);    /* 해상도 값 읽기 */

/* 메모리 할당 : 2차원 배열 */
/* 이미지의 각 행 : 높이(height)만큼 메모리 할당 */
img->pixelData = (uchar**)malloc(img->height*sizeof(uchar*));
for(x = 0; x < img->height; x++)   /* 이미지 픽셀 : 넓이(width)만큼 메모리 할당 */
    img->pixelData[x] = (uchar*)malloc(img->width*sizeof(uchar));
```


PBM 파일 읽기(6)

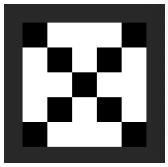
- 이미지 : x와 y축의 2차원 데이터
  → 2중 배열이나 포인터를 사용
  - 한 행을 담기 위해서는 "sizeof(unsigned char) * 가로"만큼의 메모리 필요
  - 전체 픽셀정보를 표현하려면 세로(height)의 크기만큼 반복해서 메모리를 할당

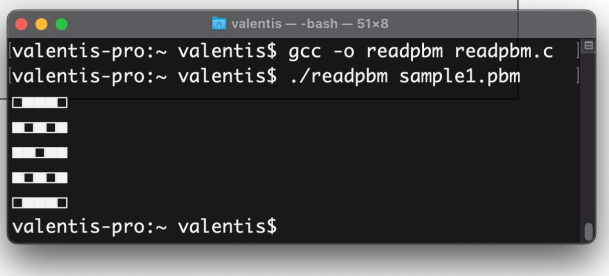



```
/* pbm 파일에서 픽셀 데이터를 읽어 할당한 메모리로 불러오기 */
for(x = 0; x < img->height; x++) {
    for(y = 0; y < img->width; y++) {
        fscanf(fp, "%u", &tmp);
        img->pixelData[x][y] = (uchar)tmp;
    }
}
```

PBM 파일 읽기(7)

```
    fclose(fp);             /* 사용이 끝난 파일은 닫는다. */

    return TRUE;
}

void releasePBM(PBMImage* img)
{
    for(int i = 0; i < img->height; i++)                    /* 2중 배열의 메모리 해제 */
        free(img->pixelData[i]);

    free(img->pixelData);
}
```





미니 프로젝트

- BPM 파일을 읽어 프레임버퍼로 출력하는 코드를 작성하시오.
  - 필요한 코드를 추가

```
typedef struct {
    char MN[2];                 /* 파일의 유형(매직넘버값) */
    int  width, height;         /* 영상의 너비   영상의 높이 */
    int max;                    /* 이미지의 최대값 */
    uchar **pixelData;          /* 이미지 데이터의 값 */
} PBMImage;
```

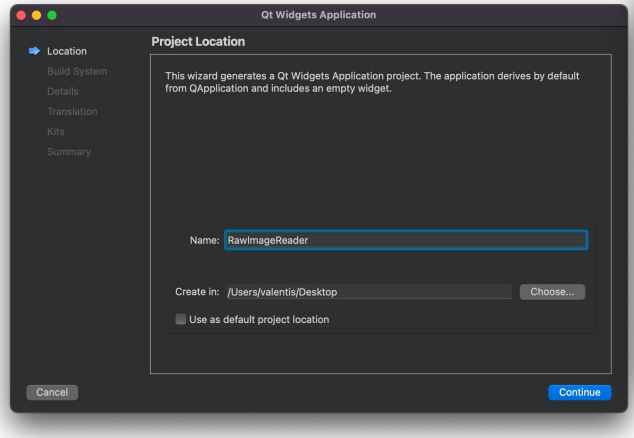
  - fscanf(fp, "%d", &img->max);   /* 최대명암도 값 */
  - 최대값으로 색상을 정규화

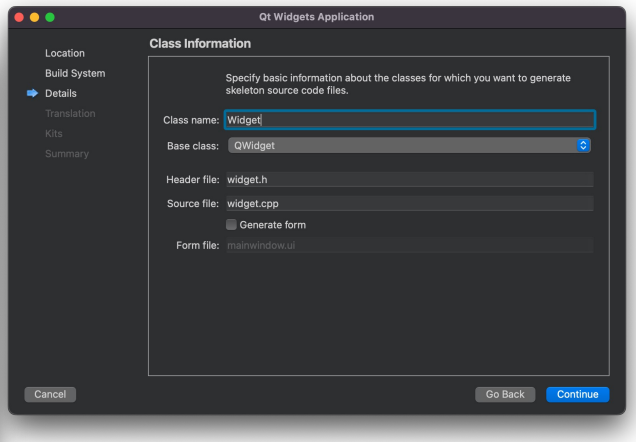
Raw Data

- 이미지 파일 포맷 중 하나로 디지털 카메라나 이미지 스케너의 이미지 센서로부터 최소한으로 처리한 데이터를 포함
- Raw Data 이미지 파일의 장점
  - 무손실 압축을 사용하므로 이미지는 원본 그대로를 사용
  - 더 섬세한 제어가 가능
  - 색 공간은 사용자가 원하는대로 변경 가능
- Raw Data 이미지 파일의 단점
  - 일반적으로 JPEG 파일보다 2~6배 정보 용량이 더 크다.
    - 카메라가 raw 이미지를 카드에 기록할 때 JPEG보다 시간이 많이 걸리므로 연속 촬영에서 건져낼 수 있는 사진 수가 비교적 적다.
  - 미리 이미지의 정보(Width, height, 색상)를 알아야 한다.

Raw Data 이미지 파일 읽기(1)

- Raw 파일 읽는 순서
  - Width와 Height와 색상을 입력
  - 이미지 데이터 읽기
- Qt Creator에서 프로젝트 생성
  - Qt Widgets Application





Raw Data 이미지 파일 읽기(2)

- main.cpp 파일만 조작

```
#include <QApplication>
#include <QWidget>
#include <QLineEdit>                    /* 해상도 입력을 위해 */
#include <QLabel>                       /* 이미지 표시 */
#include <QPushButton>                  /* 이미지 로드 동작 수행 */
#include <QBoxLayout>                   /* 위젯들을 배치 */
#include <QFileDialog>                  /* 이미지 파일 선택 */
#include <QDir>                         /* 홈디렉터리 */
#include <QFile>                        /* 이미지 파일 읽기 */

int main(int argc, char *argv[ ])
{
   QApplication a(argc, argv);          /* QApplication 객체 생성 */
   QWidget w;                           /* 메인 위젯 */
```


Raw Data 이미지 파일 읽기(3)

- QWidget 객체를 만들고 레이아웃을 이용해서 위젯을 배치
  - 해상도의 기본값은 256 x 256

```
   QLineEdit* widthLineEdit = new QLineEdit("256", &w);          /* X 해상도 입력 */
   QLineEdit* heightLineEdit = new QLineEdit("256", &w);         /* Y 해상도 입력 */
   QPushButton* button = new QPushButton("Load", &w);            /* 버튼 생성 */
   QLabel* imageLabel = new QLabel(&w);                  /* 이미지 표시를 위한 위젯 */

   QHBoxLayout *layout = new QHBoxLayout( );                     /* 위젯을 배치 */
   layout->addWidget(widthLineEdit);
   layout->addWidget(heightLineEdit);
   layout->addStretch(1);
   layout->addWidget(button);

   QVBoxLayout *mainLayout = new QVBoxLayout(&w);
   mainLayout->addLayout(layout);
   mainLayout->addWidget(imageLabel);
```

Raw Data 이미지 파일 읽기(4)

- QPushButton 위젯과 람다 함수를 이용해서 이미지 불러오기

```
QObject::connect(button, &QPushButton::clicked, [&]( ) {   /* 버튼 클릭시 동작 */
    QString fileName = QFileDialog::getOpenFileName(0,         /* 파일명 입력 */
                                "Open a file",QDir::homePath( ), "raw file(*.raw)");

    QFile file(fileName);                          /* 파일 불러오기 */
    file.open(QFile::ReadOnly);                    /* 파일 열기 */
    QByteArray byteArray = file.readAll( );    /* 파일을 읽어서 QByteArray에 저장 */
    file.close( );                                 /* 다 쓴 파일 닫기 */

    int width = widthLineEdit->text( ).toInt( );          /* 문자열을 정수로 변환 */
    int height = heightLineEdit->text( ).toInt( );
    uchar *data = (uchar*)(byteArray.data( ));    /* QByteArray를 uchar로 변환 */

    /* uchar를 QImage 객체로 변환 */
    QImage image = QImage(data, width, height, QImage::Format_Grayscale8);
```


Raw Data 이미지 파일 읽기(5)

```
    /* QImage 객체를 QLabel 위젯에 표시 */
    imageLabel->setPixmap(QPixmap::fromImage(image,Qt::AutoColor));
    /* 윈도우 제목으로 파일의 경로를 표시 */
    w.setWindowTitle(fileName);
});                        /* 람다 함수의 끝 */

w.show( );                          /* 메인 위젯을 화면에 표시 */

return a.exec( );                   /* Qt의 이벤트 루프 시작 */
}
```

- QImage 객체에서 이미지 데이터를 자동으로 처리
  - 프로그램을 시작 후 Load 버튼을 눌러서 Raw 이미지 파일을 선택하면 QLabel 위젯에 표시