고급 SW 실습 I
MFC & Digital Image
(실습자료)

CSE4152

서강대학교 컴퓨터공학과

Digital Image & Color Conversion

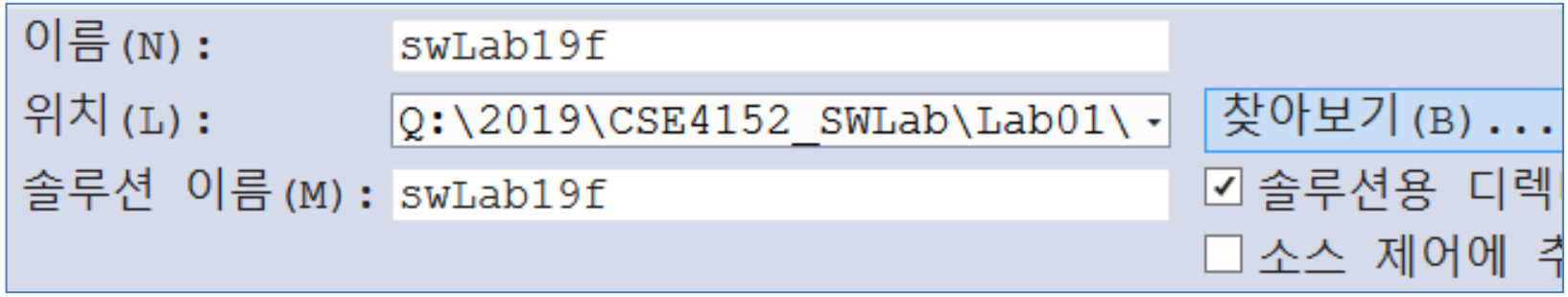
- 실습 내용
  - Visual Studio MFC를 사용한 프레임 워크 생성.
  - 프로젝트에 CDIB, OpenCV 툴킷 연결
  - 몇가지 이미지 처리 응용 프로그램 작성 및 시험.
- 주의 사항
  - 숙제와 관련이 있으니 여기 실습 내용을 정확히 작성해야 합니다.
  - 특히, 함수/변수 이름, 작성한 멤버 함수가 소속할 클래스 이름 등이 여기 실습 안내와 정확히 일치하여야 합니다.

Open CV 설치

- 구글에서 “OpenCV download” 입력 후 검색.
  - 다운로드 사이트: https://opencv.org/releases/
  - 세 종류가 있다: OpenCV 2.x.x, 3.x.x, 4.x.x
  - 기존 유저는 이중 자신이 사용하던 것을 다운 받아 설치.
  - 본 실습에서는 OpenCV 2.4.13.6 최신 버전을 다운 받자.
    - OpenCV 3xx나 4xx는 x64 라이브러리만 지원해줘 불편.
    - 다운 받는 파일(19년 가을): opencv-2.4.13.6-vc14.exe
  - 설치
    - OpenCV24136이라는 폴더를 만든다.
    - 설치 파일을 만든 폴더에 복사 후 더블 클릭.
    - 설치 종료 후 실행한 설치 파일 삭제(opencv 폴더가 생김).
    - OpenCV24136 폴더를 원하는 위치로 옮긴다(예: C:\).

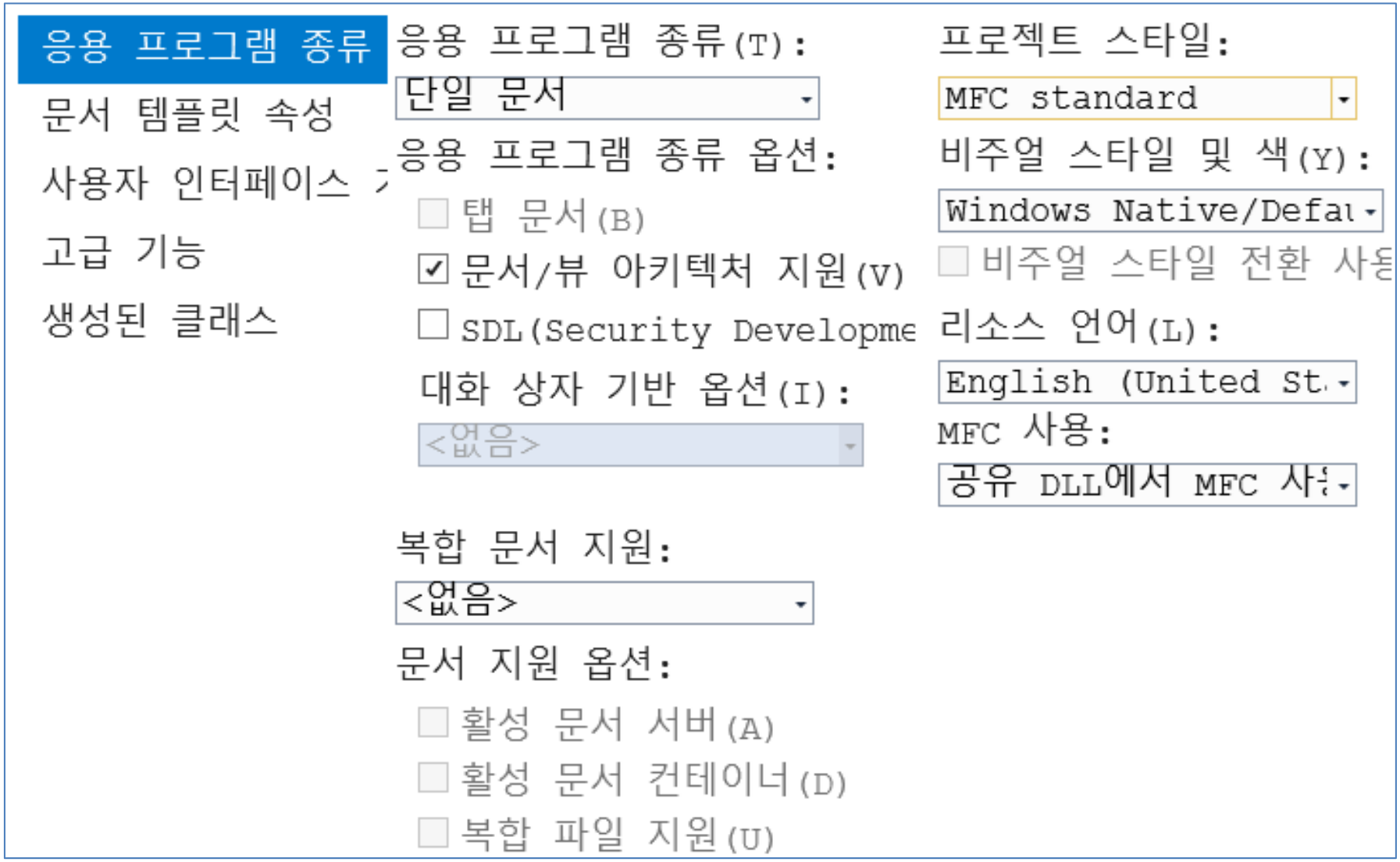
Visual Studio Project 생성(MFC SDI)

- 생성 내용
  - VS MFC는 윈도우용 응용 프로그램을 만들기에 적당하다(1).
  - MFC SDI용 프로젝트를 생성하고, 여기서 OpenCV를 사용하여 프로그램을 개발할 수 있는 환경을 만들 것이다..
  - VS2017(또는, VS2019) 실행
    - 파일 → 새로 만들기 → 프로젝트 → Visual C++ → MFC/ATL → MFC 응용 프로그램 선택
    - 아래와 같이 프로젝트 이름 입력(예: swLab19f) → 생성할 폴더 선택 → 확인

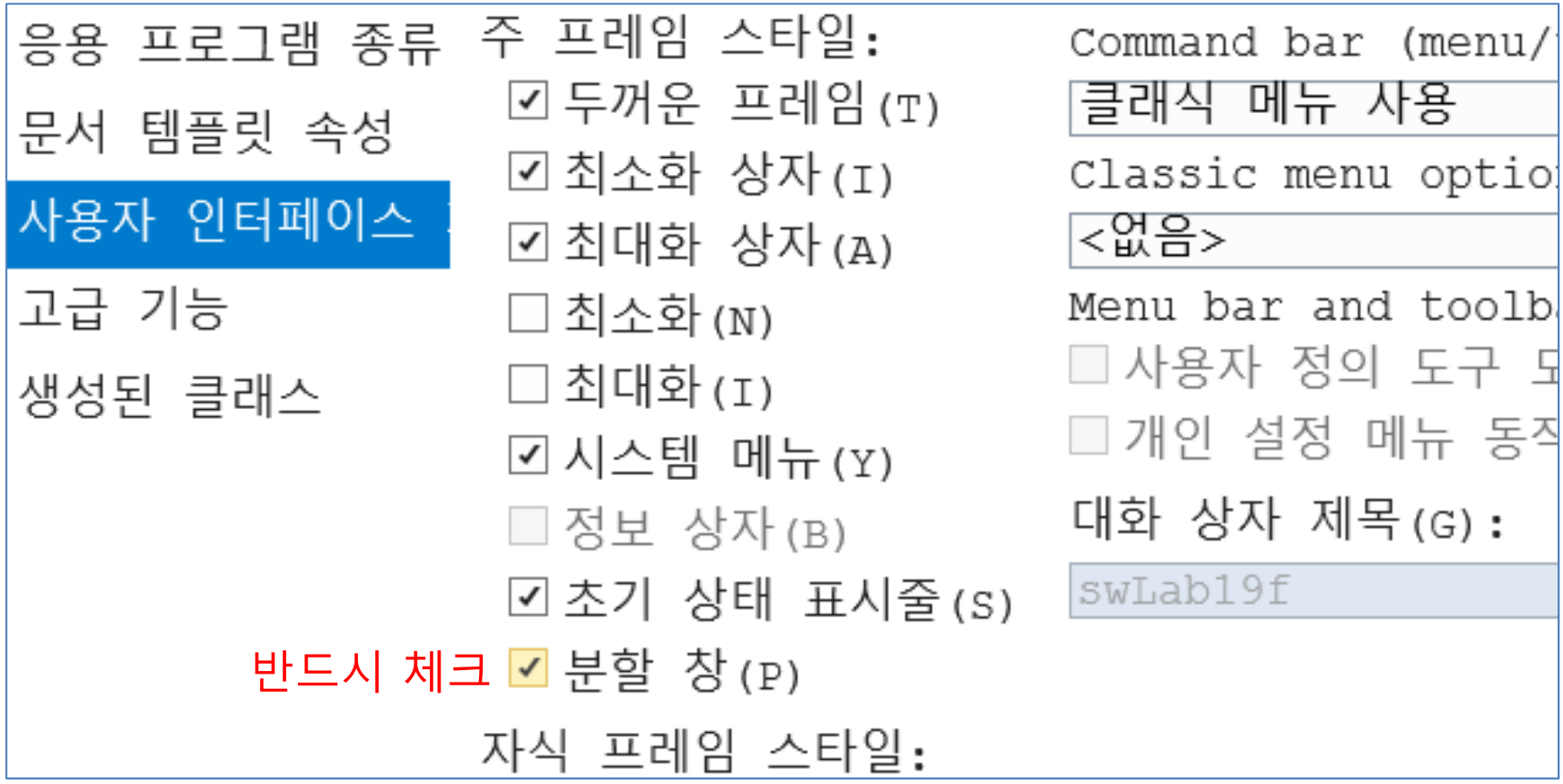


(1) 시간 될 때 다른 프레임워크도 익히자(예: Qt(https://www.qt.io/)

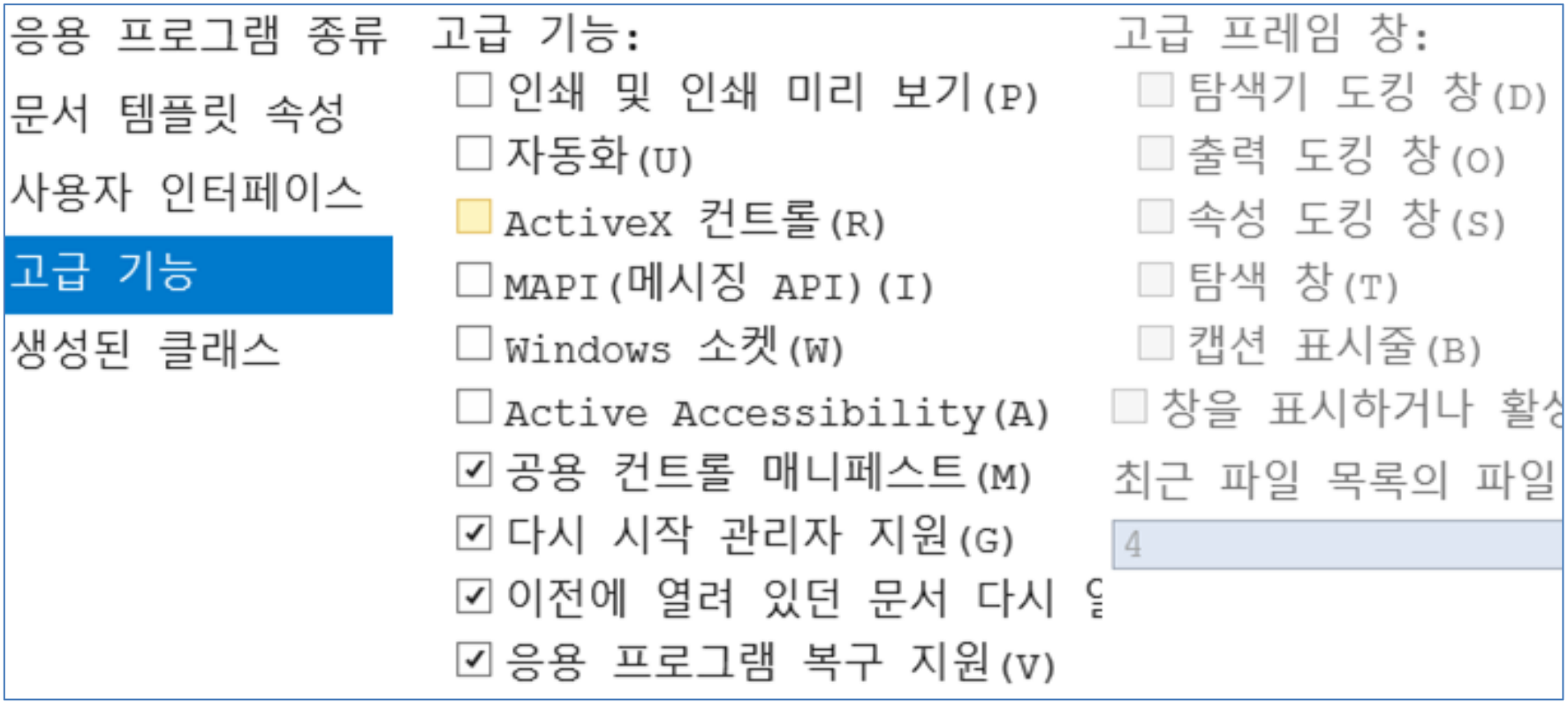
◆ 이어지는 창에서 다음과 같이 설정 → 다음



- 문서 탬플릿 속성 → 다음
- 사용자 인터페이스 기능 : 다음과 같이 설정 → 다음



- 고급 기능
  - 불필요한 것들 뺀다(인쇄, ActiveX 컨트롤 등) → 다음
- 생성된 클래스 : 마침
- 디버그 모드 x86에서 빌드하여 실행해 본다(F5)



- OpenCV 연결
  - 프로젝트 속성 열고 Debug, 활성(Win32)선택
  - C/C++ → 일반 → 추가 포함 디렉토리

    C:\OpenCV24136\opencv\build\include; C:\OpenCV24136\opencv\build\include\opencv;%(AdditionalIncludeDirectories) 추가(창에서 해당 폴더로 가서 경로 copy/paste)
  - 링커 → 일반 → 추가 라이브러리 디렉토리

    C:\OpenCV24136\opencv\build\x86\vc14\lib;%(AdditionalLibraryDirectories) 추가
  - 링커 → 입력 → 추가 종속성

    opencv_core2413d.lib; opencv_highgui2413d.lib; 추가(기타 필요한 것이 있으면 차 후 추가).
  - 디버깅 → 환경

    PATH= C:\OpenCV24136\opencv\build\x86\vc14\bin;%PATH% 추가.
  - 적용

- 메모리 누수: 디버그 모드에서 실행하면 메모리 누수(leak) 현상이 발생한다.

  이를 방지하기 위하여 프로젝트 속성 → 링커 → 입력 → 지연 로드된 DLL 항목에 opencv_core2413d.dll;을 추가(1).

- 속성을 릴리즈 모드로 바꾸고 같은 작업

  단 링커 → 입력 → 추가 종속성에

  opencv_core2413.lib;opencv_highgui2413.lib; 추가

- x64 빌드를 원한다면,
  - 프로젝트 속성 → (디버그/릴리즈 각각, x64) → x86과 동일하게 설정하되, 링커 → 일반 → 추가 라이브러리 디렉토리를 C:\OpenCV24136\opencv\build\x64\vc14\lib으로 설정

(1) 확실한 대처인지 모름.

- 시스템 경로 설정
  - VS2017을 통하지 않고 직접 실행할 때 필요하다(그러나, OpenCV가 없는 컴퓨터에서는 필요한 dll을 같은 폴더에 두어야 실행된다).
  - 내컴퓨터 우클릭 → 속성 → 고급 시스템 설정 → 환경 변수 → 사용자 변수에서 Path를 찾아 클릭 → 편집 → 새로 만들기 클릭 후

    C:\OpenCV24136\opencv\build\x86\vc14\bin 추가.
  - X64에서 실행시키기를 원할 경우
    - 경로 C:\OpenCV24136\opencv\build\x64\vc14\bin를 마저 추가.

테스트 실행

- snnnnnnL01_Mfc.cpp 파일을 하나 만들어 프로젝트에 등록 (nnnnnn은 자신의 학번 뒤 6자리, 자료에서 nnnnnn 표시는 모두 이렇게 학번으로 바꾼다)

  앞으로 모든 MFC 입출력 관련 기본 함수는 모두 이 파일에 저장한다.

- 다음과 같은 코드를 넣어 경로 설정 체크

  오류가 생기면 앞쪽 경로 다시 확인

```
#include "stdafx.h"
#include <opencv2/core.hpp>
void foo(void) {
}
```

- 생성된 클래스
  - 프로젝트를 생성하면 다음과 같이 네 개의 클래스가 생긴다:
    - CMainFrame, C….App, C….Doc, C….View
    - 우리의 경우, CMainFrame, CswLab19fApp, CswLab19fDoc, CswLab19fView 인데 이에 대한 ~.cpp와 ~.h 파일이 솔루션 탐색기에 보일 것이다.
    - 여기서, CswLab19fApp는 swLab19f.cpp (~.h)에 대응된다.
  - 이들 클래스에 우리의 응용을 위한 멤버 함수를 적절히 추가하여, 원하는 프로그램을 만드는데, 이 경우 프로그램이 지저분(1) 해질 수 있다.
  - 따라서, 모든 응용 프로그램은 함수로 작성하고, 클래스에서는 이들 함수를 호출만 하도록 한다.
  - 즉, 작성할 프로그램 대부분을 snnnnnnL01_MFC.cpp와 같은 몇 개의 파일에 저장하여 관리를 편하게 한다.

(1) 관리 불편, 이해하기도 어렵고...

화면 나누기

- 계산 시간, 입출력 정보 등을 화면에 출력할 필요가 있다.
- 이를 위하여 View 창을 상하 둘로 나누어 상부는 이미지 출력에, 하부는 문자 출력에 사용할 것이다.
- 창 분할을 위하여 처음 프로젝트 생성시, 사용자 인터페이스 기능에서 분할 창을 체크하였음을 기억하자.
- 이 결과로 CMainFrame의 멤버 함수 OnCreateClient을 보면 다음과 같은 코드가 있다(1)(MainFrame.cpp).

```
return m_wndSplitter.Create(this,
    2, 2,                 // TODO: 행 및 열의 개수를 조정합니다.
    CSize(10, 10),        // TODO: 최소 창 크기를 조정합니다.
    pContext);
```

- 이 부분을 우리의 목적에 맞게 첨부한 Code01.txt의 내용으로 바꾼다.

(1) 만일 없다면, 분할 창을 체크하지 않은 것이니, 프로젝트를 다시 생성해야 한다.

✦ 대체할 코드는 다음과 같다:

```
BOOL flag = m_wndSplitter.CreateStatic(this, 2, 1);     // 창을 2 x 1 로 나눈다
if (flag) {     // *** 각 창별로 뷰 클래스를 지정한다
     m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CswLab19fView),
          CSize(100, 100), pContext);  // 위 창은 C…View 창으로 사용(그리기 창)
     m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CRichEditView),
          CSize(100, 100), pContext);  // 아래 창은 문자 출력용으로 사용
  initMessage((CRichEditView*)m_wndSplitter.GetPane(1, 0));
  m_bSplitter = true;   // 분할되었음을 알리는 flag
}
return flag;
```

✦ CRichEditView는 문서 처리 기능을 갖는 클래스인데, 이를 통하여 문자열 출력 함수가 있는 MsgView.cpp (~.h) 두 파일을 프로젝트에 복사, 등록(1)하고, 다음 코드를 추가한다.

MainFrm.cpp의 #include "MainFrm.h" 다음에

```
#include "swLab19fDoc.h"
#include "swLab19fView.h"
#include "MsgView.h"
```

MainFrm.h에 protected 멤버 변수로

```
bool m_bSplitter = false;
```

(1) 별첨 코드에 있다.

- 창 크기가 바뀌었을 때를 위한 대처
  - 별첨 코드의 Code02..txt를 MainFrm.cpp의 마지막 부분에 추가한다(창 크기 변화시 호출되는 함수).
  - MainFrm.h의 마지막 부분에 CMainFrame의 메시지 맵 함수로 아래와 같이 추가하고

```
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
afx_msg void OnSize(UINT nType, int cx, int cy);  // 추가
DECLARE_MESSAGE_MAP( )
```

  - MainFrm.cpp의 메시지 맵에 아래와 같이 매칭시킨 후

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        ON_WM_CREATE( )
        ON_WM_SIZE( )    // 추가
END_MESSAGE_MAP( )
```

  - swLab19fView.cpp에 다음과 같이 등록 후 빌드.

```
BEGIN_MESSAGE_MAP(CswLab19fView, CView)
        ON_WM_SIZE( )    // 추가
END_MESSAGE_MAP()
```

- ✦코드에서 ON_WM_SIZE( )는 윈도우에 변화가 있을 때 (View 또는 Frame) 발생하는 메시지인데 이를 메시지 맵에서 대응되는 함수를 찾아 이를 호출한다(OnSize( )).
- ✦함수 OnSize( )는 View를 항상 8 (이미지) : 2 (문자열)로 나누어 주는 기능을 수행한다(물론 마우스로도 조정할 수 있다).

◆ showMessage( ) (MsgView.cpp)

```
void showMessage(CString pszMsg)
```

- ✦CString은 MFC에서 제공하는 문자열 클래스이다(1).
- ✦인수를 문자열로 할 경우 L"…."로 앞에 L을 붙인다.
- ✦.Format( )이라는 편리한 멤버 함수가 있다.

◆ 이제 빌드해보자. 문제가 없어야 한다.

(1) https://docs.microsoft.com/ko-kr/cpp/atl-mfc-shared/using-cstring?view=vs-2019

프로그램 작성

- 프로그래밍 전략
  - snnnnnnL01.h
    - 여기에는 학생 자신의 클래스를 정의한다(멤버 변수 및 멤버 함수(일단 모두 public으로 한 후, 차후 protected/private로 해도 되는 것은 protected/private로 옮기자).
    - 클래스 이름은 편의상 SWL01로 한다.
  - snnnnnnL01_Mfc.cpp
    - 클래스 인스턴스를 위한 전역 변수 선언
    - SWL01의 멤버 함수 코드(주로 입출력 함수)
  - snnnnnnL01_ext.h
    - snnnnnnL01_Mfc.cpp에서 정의한 전역 변수의 extern.
  - snnnnnnL01_App.cpp
    - SWL01의 멤버 함수 코드(주로 응용 함수)
  - MFC 클래스 내에서는 SWL01 클래스의 함수 호출에 필요한 최소한의 코딩만 한다.

- 작성 예
  - 이미지 입력 기능 작성할 경우
    - 리소스 편집기를 사용하여 메뉴 편집 후 ID 수정(필요시) 및 이벤트 처리기를 적절한 MFC 클래스에 추가.
    - 이미지 읽는 멤버 함수를 예로 들면
      - snnnnnnL01_Mfc.h에 필요한 멤버 변수 및 함수 정의.
      - snnnnnnL01_Mfc.cpp에 멤버 함수 작성.
    - 이벤트 처리기에서 멤버 함수 호출

      이때, SWL01 클래스 인스턴스가 필요하므로, 전역 변수로 snnnnnnL01_Mfc.cpp 에 하나 정의하고,

      snnnnnnL01_ext.h에 extern 변수로 추가한다.

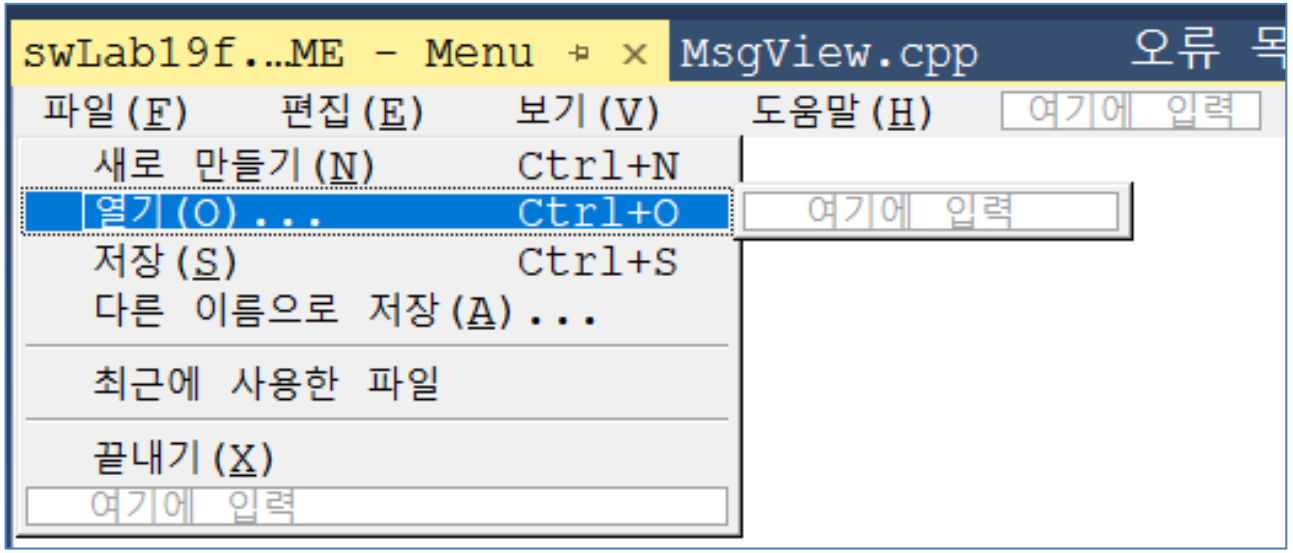
이미지를 읽어 창에 보이기

파일 읽기

- 이제 이미지 파일을 읽어 내부 자료에 저장하는 프로그램을 작성하자.
- 먼저, 별첨 코드의 CDIB.cpp와 CDIB.h를 프로젝트에 복사하고 등록한다.
- CDIB.cpp에는 .bmp 파일을 다루는 함수들이 작성되어 있다(출처?).
- OpenCV와 같은 도구의 함수를 이용해도 되지만, 16 bit 컬러 이미지와 같은 경우에는 제대로 처리가 안되어, .bmp 내용을 정확히 그대로 읽을 수 있는 CDIB.cpp 함수를 사용한다.

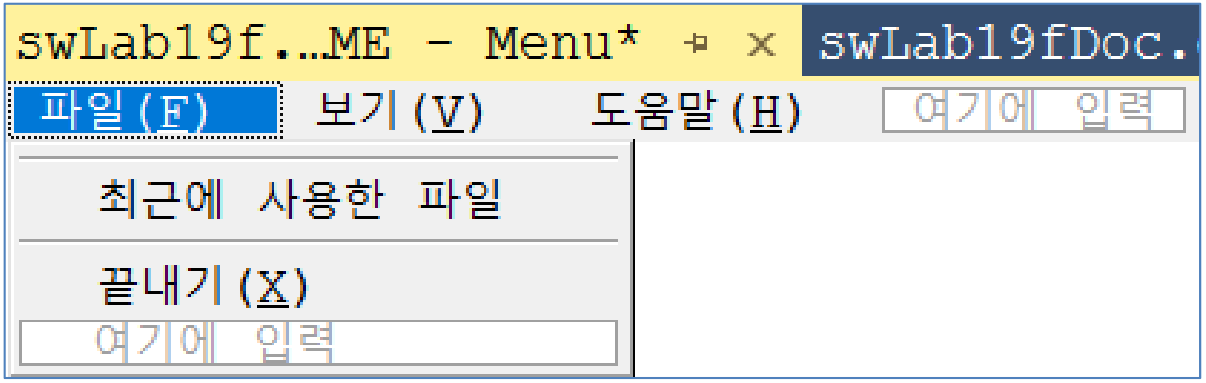
메뉴 정리 및 수정

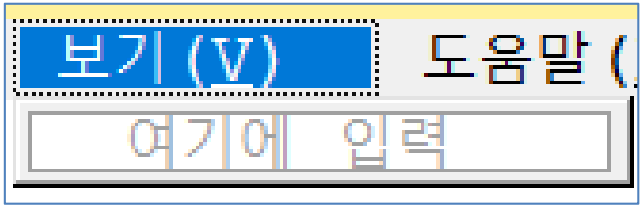
- 솔루션 탐색기 → 리소스 파일 → swLab19f.rc 더블 클릭 → Menu → IDR_MAINFRAME 더블 클릭 → 메뉴 편집 창



- 여기에 입력 칸에 메뉴 추가 가능.
- 이미 있는 메뉴에 대해 마우스 클릭 후
  - 다시 클릭하면 caption을 수정할 수 있고,
  - 드래그하면 자리를 바꿀 수 있으며,
  - 우클릭 하면 삭제, (속성 선택하여) ID 편집, 이벤트 처리기 추가 등의 작업을 할 수 있다.

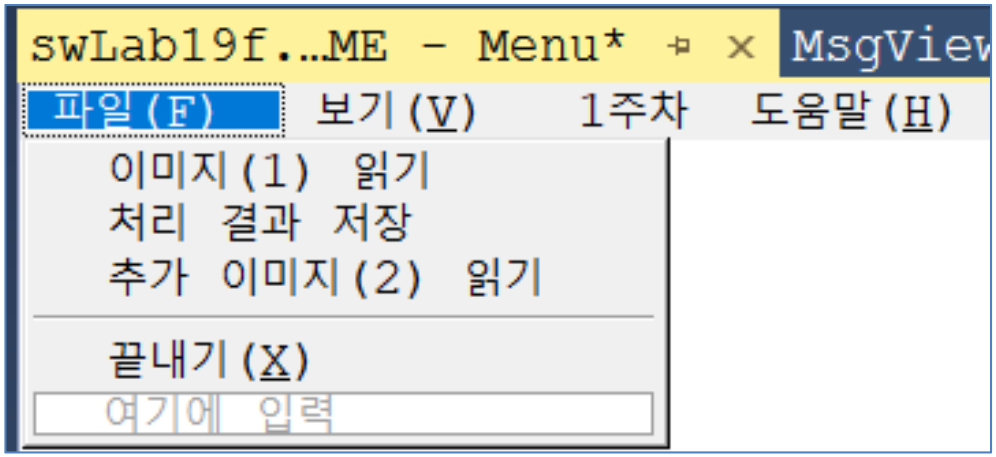
- 메뉴에 있는 새로 만들기, 열기 등은 이미 ID가 배정되고 멤버 함수만 만들면 된다.
- 그러나 이들은 CswLab19fDoc 클래스의 멤버 함수 Serialize( )와 이미 연결되어 맘대로 가로채서 사용하기 불편하다(1).
- 따라서, 이를 무시하고(즉, 모든 메뉴를 일단 지우고), 우리의 목적에 맞게 메뉴를 수정하도록 한다.
- 다음과 같이 메뉴를 모두 지운 후(2) 필요한 것을 추가하자.



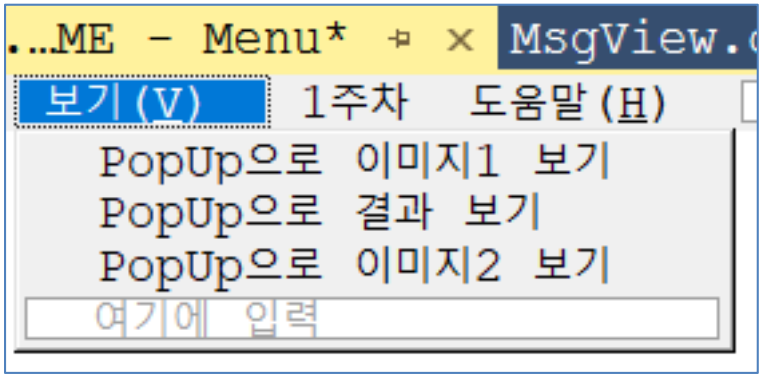


(1) 흥미 있는 학생은 차 후 CArchive 클래스에 대해 공부해 보도록 하자.
(2) 그렇다고 이미 있던 코드는 지워지지 않는다. 단지 안쓸 뿐이다. 즉, 메뉴에 이벤트 처리기를 붙였을 경우, 이 메뉴를 지워도 이벤트 처리기와 관련 코드는 그대로 남아 있다(수동으로 일일이 찾아서 지워야 한다. 신중히 작업하자.)

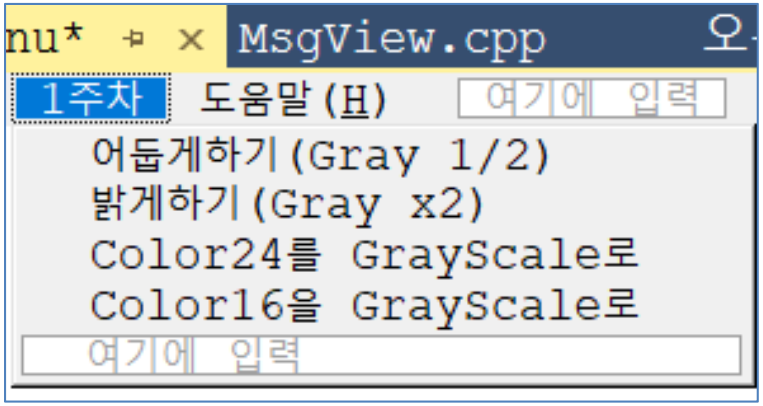
- 아래와 같은 형태로 메뉴를 추가하자(완료, 확인 후 저장)
  - 파일 메뉴:



  - 보기 메뉴:



  - 1주차 메뉴:



- 메뉴 처리기 추가
  - 이미지(1) 읽기를 우 클릭하여 속성을 본다.
  - ID를 ID_FOPEN01로 수정한다(1).
    - 다시 속성을 보고 확인해야 한다(수정이 안될 수 있다).
    - 다른 메뉴를 처리할 때도 알기 쉬운 ID로 수정한다.
  - 이미지(1) 읽기를 다시 우 클릭 → 이벤트 처리기 추가 선택
    - 메시지 형식: COMMAND
    - 함수 처리기 이름: On_FOpen1
    - 클래스는 CswLab19fDoc를 선택(사실 MFC 클래스는 어떤 것이든 선택 가능. 다른 클래스는 선택 불가).
    - 추가 및 편집 클릭 → swLab19fDoc.cpp에 함수가 생기고 코드 추가가 가능해 진다.
    - 일단, 모두 저장을 클릭하고 다음 쪽으로…

(1) 작명을 잘 하자. 이후 프로그램을 검토하거나 이해할 때 필요하다.

✦ 이미지(1) 읽기에 대응하는 이벤트 처리

◆ swLab19fDoc.cpp에 이미 아래 보인 코드가 추가되어 있다.

```
…
// ID가 ID_FOPEN01인 메시지가 발생하면 On_FOpen1을 호출
BEGIN_MESSAGE_MAP(CswLab19fDoc, CDocument)
        ON_COMMAND(ID_FOPEN01, &CswLab19fDoc::On_FOpen1)
END_MESSAGE_MAP()
…
…
void CswLab19fDoc::On_FOpen1( )
{
        // TODO: 여기에 명령 처리기 코드를 추가합니다.
}
```

◆ swLab19fDoc.h에도 On_FOpen1( )이 이미 선언되어 있다.

◆ 이들을 확인 후 빌드 해서 문제가 없는지 확인한다.

◆ 파일 편집

✦함수 OnFileOpen1( )에 별첨 코드 Code03..txt를 추가한다.

✦이 코드는 파일 선택창을 통하여 파일을 선택하고 이의 경로 및 파일 이름을 pathName에 저장한다.
여기서, SWL01_inst는 SWL01 클래스의 인스턴스로 s074419L01_Mfc.cpp에 광역 변수로 선언할 것이다.

✦SWL01_inst은 우리의 클래스 SWL01에 대한 instance이다(1).
아직 이를 설정하지 않았으므로 오류 밑줄이 보일 것이다.

```
CFileDialog dlgfile(TRUE);
// show file open dialog
if (IDOK == dlgfile.DoModal()) {
  CString pathName = dlgfile.GetPathName();
    // image 1을 읽기에 아래 인수는 1 (나중에 설명)
  SWL01_inst.readImage(pathName, 1);
}
```

(1) 클래스 밖에서 멤버 함수를 호출하려면 instance가 반드시 필요하다.

- 이제 다음과 같은 우리의 파일들을 편집해야 한다:

```
snnnnnnL01.h
snnnnnnL01_Mfc.cpp
snnnnnnL01_ext.h
```

- 일단 이들을 빈 채로 만들어 솔루션 탐색기에 등록하자(1).
- 그리고, CDIB.cpp와 CDIB.h 등록을 확인하자.

(1) snnnnnnL01_Mfc.cpp는 전에 이미 등록하였다.

snnnnnnL01.h

✦우리의 클래스 SWL01 정의(Code04..txt의 내용을 복사).

```
#include <opencv2/opencv.hpp>   // OpenCV 헤더
#include "MsgView.h" // 문자열 출력 함수(CString)
#include "CDIB.h"    // ~.bmp 처리 모듈 헤더

using namespace std; // namespace std:: 생략
using namespace cv;  //      OpenCV cv:: 생략

class SWL01 {
protected:
  // DIB data(image1) (read from a .bmp file)
  Cdib     m_dibFile1;
  LONG     m_width1, m_height1; // height and width
  int      m_depth1;            // bits per pixel
  LPBITMAPINFO mg_lpBMIH1;      // bmp infoHeader

  // DIB data(image2)
  ...
```

snnnnnnL01.h (계속)

```
public:
  bool readImageF1;      // image 1 읽으면 true
  bool readImageF2;      // image 2 읽으면 true
  bool processed;   // 뭐든 image 처리를 마치면 true
  SWL01( ) {         // constructor
    readImageF1 = false;
    readImageF2 = false;
    processedF = false;
  }
  ~SWL01() {         // destructor
  }
  // image read function(which=1(image1), 2(image2),
  //                     3(처리 결과 image(차후 설정))
  void readImage(CString pathName, int which);

  void drawImage(CDC *pDC, int dcLTx, int dcLTy,
                 int which);  // image display 함수
};
```

snnnnnnL01_Mfc.cpp

✦멤버 함수 추가 (Code05.txt의 내용을 복사).

```
#include "s074419L01.h"
#include "CDIB.h"

SWL01   SWL01_inst;        // SWL01 instance
CswLab19fView   *g_pView = NULL;   // View instance
CMainFrame      *g_pMainF = NULL; // Frame instant

void SWL01::readImage(CString pathName, int which) {
// read an image & store it to image1 or image2
  CFile    file;    // CFile class(C에서 FILE과 유사)
  LPCTSTR str = pathName;  // CString과 유사한 class
  // 아래 모두 포인터로 한 이유는 두 이미지를 읽어야 해서
  CDib     *pDibF;           // DIB class
  LONG     *width, *height;
  int      *depth;
  LPBITMAPINFO *bmInfoHd;   // DIB infoHeader
```

snnnnnnL01_Mfc.cpp (계속)

Pointer setting

```
if (which == 1) {
  pDibF = &m_dibFile1;    // reading image1
  width = &m_width1;  height = &m_height1;
  depth = &m_depth1;  bmInfoHd = &mg_lpBMIH1;
  readImageF1 = true;
}
else if (which == 2) {
  if (readImageF1 == false) {
    AfxMessageBox(L"Read image1 first!", MB_OK, 0);
    return;  // image2는 image1 읽은 후에나 읽을 수 있다
  }
  pDibF = &m_dibFile2;  // when reading image2
  width = &m_width2; height = &m_height2;
  depth = &m_depth2;  bmInfoHd = &mg_lpBMIH2;
  readImageF2 = true;
}
```

snnnnnnL01_Mfc.cpp (계속)

Image reading

```
(*pDibF).Empty();  // clear any previouse read image
file.Open(str, CFile::modeRead);
if ((*pDibF).Read(&file) != TRUE) {
  file.Close();
  return;
}
file.Close();

*width = ((*pDibF).m_lpBMIH)->biWidth;
*height = ((*pDibF).m_lpBMIH)->biHeight;
*depth = ((*pDibF).m_lpBMIH)->biBitCount;
*bmInfoHd = (LPBITMAPINFO)((*pDibF).m_lpBMIH);

processedF = false;      // true after doing something
```

snnnnnnL01_Mfc.cpp (계속)

Message 출력 및 그리기 요청

```
  CString msga, msgb;

  msga.Format(L"Read an image from %s and save it to
              image %d", // 윗 줄과 같은 줄
    pathName, which);

  msgb.Format( L"  Its size and depth are
     (w = %d, h = %d, depth = %d)", // 윗 줄과 같은 줄
    *width, *height, *depth);

  showMessage(msga);
  showMessage(msgb);

  g_pView->Invalidate();  // 그리기 요청
}
```

snnnnnnL01_Mfc.cpp (계속)

이미지 그리는 함수 (OnDraw에서 호출)

```
void SWL01::drawImage(CDC *pDC,
                int dcLTx, int dcLTy, int which) {
  if (which == 1) {
    m_dibFile1.Draw( pDC,
        CPoint(dcLTx, dcLTy),
        CSize(m_width1, m_height1) );
  }
  else if (which == 2) {
    m_dibFile2.Draw( pDC,
       CPoint(dcLTx + 20 + m_width1, dcLTy),
       CSize(m_width2, m_height2) );
  }
}
```

snnnnnnL01_ext.h

- snnnnnnL01_Mfc.cpp에서 정의한 global 변수들을 extern으로 선언한다.
- 별첨 코드의 Code06.txt의 내용을 이 파일에 추가한다.

```
extern SWL01SWL01_inst;
extern CswLab19fView*g_pView;
extern CMainFrame*g_pMainF;
```

Include 파일 추가

- 네 파일 MainFrm.cpp, swLab19f.cpp, swLab19fDoc.cpp와 swLab19fView.cpp에 각각 다음과 같은 파일들이 include 되어 있는지 확인하고, 안되어 있다면 추가한다.

```
#include "swLab19fDoc.h"
#include "swLab19fView.h"
#include "MainFrm.h"
#include "snnnnnnL01.h"
#include "snnnnnnL01_ext.h"
```

- 파일 Code06_ext 헤더.txt에 위 내용이 있으니 이를 복사하자.
- nnnnnn을 자신의 학번 뒤 6자리로 바꾸는 것을 잊지 말자,

- 각 클래스별 instance 얻기
  - 클래스 밖에서 현재 활성화된 instance가 필요할 수 있다.
  - 클래스 CswLab19fView : 파일 swLab19fView.cpp에서
    - 함수 PreCreateWindow( )에 다음과 같이 추가한다.

```
BOOL CswLab19fView::PreCreateWindowlCREATESTRUCT& cs)
{
    // TODO: CREATESTRUCT cs를 수정하여 여기에서
    //  Window 클래스 또는 스타일을 수정합니다.
    g_pView = this;
    return CView::PreCreateWindow(cs);
}
```

  - 클래스 CMainFrame : 파일 MainFrame.cpp에서
    - 함수 OnCreate( )의 마지막 return 0전에 다음 코드를 추가

```
g_pMainF = this;
```

◆ Drawing 함수 추가

✦swLab19fView.cpp 의 함수 OnDraw( ) 를 별첨 코드 Code07.txt의 내용으로 바꾼다.

```
void CswLab19fView::OnDraw(CDC* pDC)
{
  CswLab19fDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (!pDoc)
    return;

  // TODO: 여기에 원시 데이터에 대한 그리기 코드를 ...
    if (SWL01_inst.readImageF1 == true) {
      SWL01_inst.DrawImage(pDC, 20, 20, 1);
    }
    if (SWL01_inst.readImageF2 == true) {
      SWL01_inst.DrawImage(pDC, 20, 20, 2);
    }
}
```

- 이제 빌드 후 실행하여 적당한 .bmp 파일을 읽어 보자.
  - 화면 좌측에 읽은 이미지가 보일 것이다.
  - 화면 아래에는 읽은 이미지의 특성(width, height, bits/pixel)이 보일 것이다.

연습

- 추가 이미지 읽는 기능을 추가해보자.
- 아주 간단하다.
  - 메뉴 편집기에서 추가 이미지(2) 읽기 수정
    - ID : ID_FOPEN02
    - 이벤트 편집기 추가에서
      - 메시지 형식: COMMAND
      - 클래스: CswLab19fDoc
      - 처리기 이름: On_FOpen2
  - swLab19fDoc.cpp에서 함수 On_FOpen2( ) 작성
    - 함수 On_FOpen1( )과 파라미터 값 만 다르고 동일.
- 완료, 빌드 후 바로 추가 이미지(2)를 읽어보자 → 오류 창.
- 첫번째 이미지를 읽은 후 추가 이미지를 읽어보자.

cv:: Mat(1)

- cv::Mat
  - 현재 읽은 이미지는 클래스 CDib instance인 m_dibFile1과 m_dibFile1에 저장 되어 있다.
  - 이 이미지 데이터를 OpenCV 클래스 Mat으로 변환하여 목적에 맞게 처리하면 대단히 편리하다.
  - 클래스 Mat은 이미지 뿐만 아니라 거의 모든 데이터 형식을 저장/ 처리 할 수 있다.
  - Mat을 대상으로 이미지/동영상 처리, 행렬 연산 등 다양한 함수를 제공하고 있어 관련 프로그램을 짧게 간단히 작성할 수 있다.

(1) https://docs.opencv.org/4.1.1/d3/d63/classcv_1_1Mat.html

Mat 멤버 변수 및 함수

- Mat 배열을 생성하는 기본적인 방법은 다음과 같다:

```
cv::Mat m = Mat(height, width, type);
```

여기서, type은 Mat 배열 원소의 데이터 타입이다.

- 이어지는 과정에 필요한 멤버 변수/함수는 다음과 같다:
  - m.rows : height
  - m.cols : width
  - m.data : 실제 배열 데이터 시작 위치(포인터)
  - m.channels( ) : 배열 원소의 채널 수.
  - m.release( ) : Mat를 deallocation

✦Mat.type( )

◆Mat 배열 원소의 유형을 사전 정의한 수로 반환한다.

	C1	C2	C3	C4	C(5)	C(6)	C(7)	C(8)
CV_8U	0	8	16	24	32	40	48	56
CV_8S	1	9	17	25	33	41	49	57
CV_16U	2	10	18	26	34	42	50	58
CV_16S	3	11	19	27	35	43	51	59
CV_32S	4	12	20	28	36	44	52	60
CV_32F	5	13	21	29	37	45	53	61
CV_64F	6	14	22	30	38	46	54	62

U : unsigned, S : signed, F : float, C : 채널 수

예 : CV_8UC1 : 8 bit unsigned char 1 채널, 즉, 8 bit grayscale
(8 bit unsigned char는 uchar로 정의되어 있다)

CV_8UC3 : 8 bit unsigned char 3 채널, 즉, 24 bit color.

CV_32FC1 : 32 bit float 1 채널

CV_64FC3 : 64 bit double 3 채널

- 주어진 Mat의 type을 심볼로 출력하는 함수를 Code8.txt에 주어졌다.
- 이를 자신의 프로그램에 클래스 외 함수로 넣어두자.
  - s074419L01_Mfc.cpp에 Code8.txt의 내용을 복사한다.
  - s074419L01.h의 SWL01 클래스 정의 전에 다음과 같이 선언한다:

```
string type2str(int type);
```

Mat 데이터 읽기, 쓰기

Mat::at

- M.at<pixel type>(r,c)[channel] 형태로 접근
- 단일 채널일 경우 이를 생략 한다 (.at<uchar>(r,c))
- 다음 쪽에서 설명하는 두 방법에 비해 속도가 느리다.
- CDib 이미지 데이터를 Mat 이미지로 복사하는 예

```
// h : height, w : width, type = CV_8UC3라고 가정
// DibImg : CDib 24bit color 이미지라고 가정
unsigned char *pDibData = DibImg.m_lpImage;
Mat M = Mat(h, w, CV_8UC3);
for (int r = 0; r < h; r++)
  for (int c = 0; c < w; c++)
    for(int ch=0; ch<(*pMat).channels(); ch++)
      M.at<Vec3b>(r, c)[ch] = *pDibData++;
```

✦Mat::ptr

◆M.ptr<pixel type>(row)

- 주어진 행 row의 시작 pointer를 얻을 수 있다.
- 예 (row by row로 처리하므로 속도가 .at 보다 빠르다)

```
// h : height, w : width, type = CV_8UC3라고 가정
// DibImg : CDib 24bit color 이미지라고 가정
unsigned char *pDibData = DibImg.m_lpImage;
Mat M = Mat(h, w, CV_8UC3);
for (int r = 0; r < h; r++)
  uchar *pMatRow = M.ptr<uchar>(r);
  for (int c = 0; c < w; c++) {
    pMatRow[3 * c    ] = *pDib_data++;
    pMatRow[3 * c + 1] = *pDib_data++;
    pMatRow[3 * c + 2] = *pDib_data++;
  }
```

✦Mat::data

◆M.data의 포인터를 직접 얻어 복사

- 이미지 데이터의 시작 pointer를 얻어 복사한다.
- 예 (속도가 보다 빠를 것이다)

```
// h : height, w : width, type = CV_8UC3라고 가정
// DibImg : CDib 24bit color 이미지라고 가정
unsigned char *pDibData = DibImg.m_lpImage;
Mat M = Mat(h, w, CV_8UC3);
uchar *pMatData = (*pMat).data;
for (int r = 0; r < h; r++)
  for (int c = 0; c < w; c++) {
    pMatData[r * h * 3 + c * 3    ] = *pDib_data++;
    pMatData[r * h * 3 + c * 3 + 1] = *pDib_data++;
    pMatData[r * h * 3 + c * 3 + 2] = *pDib_data++;
  }
```

✦Mat::data (계속)

- ◆포인터 연산을 사용한다.

```
for (int r = 0; r < h; r++)
  for (int c = 0; c < w; c++) {
    *pMatData++ = *pDib_data++;
    *pMatData++ = *pDib_data++;
    *pMatData++ = *pDib_data++;
  }
```

- ◆Single loop로도 가능한다 (가장 빠를 것)

```
for (int p = 0; p < h * w * M.channels(); p++) {
  *pMatData++ = *pDib_data++;
}
```

PopUp 창으로 이미지 보기

- PopUp 창으로 이미지 보기
  - OpenCV에서 함수를 사용하여, 이미지를 popup 창으로 볼 수 있다:

```
void namedWindow(const string& winname, int flags)
 winname : 윈도우 이름
 flags : WINDOW_AUTOSIZE(이미지 크기와 동일, default)
         WINDOW_NORMAL(크기 조정 가능) 등.

void imshow(const string& winname, CvArr* mat)
 winname : 윈도우 이름(namedWindow에서 지정한 이름과
   동일하게 설정. Autosize인 경우 단독 호출로도 충분)
 mat : OpenCV에서 정의한 다목적 배열(Mat, IplImage 등)

int waitKey(int delay)
 delay 만큼 지연(= 0이면 무한 대기(창을 닫을 때까지 유지)).
```

  - 이제 PopUp으로 이미지1 보기 메뉴를 프로그래밍하자.

CDib 이미지를 Mat 으로 변환

- PopUp 창으로 이미지를 보기 위해서는 CDib 이미지를 Mat 으로 변환해야 한다.
- 파일 snnnnnnL01.h를 별첨 코드의 Code09.txt로 바꾼다.
  - 추가한 Mat
    - M_Mat1 (m_Mat2) : 이미지 1 (2)에 해당하는 Mat.
    - M_MatR : 이미지 처리 결과를 저장하는 Mat(차후 사용).
  - 추가한 함수
    - DIBtoMat(which=1|2) : CDib 를 Mat 으로 변환.
    - DrawMatPopUp(which=1|2|3) : Mat을 PopUp 창에 보이기.
      - which = 1, 2는 각각 읽은 이미지 1과 2를 의미하고, 3은 차후 이미지 처리 결과를 보일 때 사용한다.
    - string type2str(int type) : Mat.type( ) 결과를 심볼로 반환.

- s074419L01_Mfc.cpp
  - Code10.txt의 내용으로 바꾼다.
  - 멤버 함수 DIBtoMat과 drawMatPopUp를 추가 하였다.
  - 멤버 함수 readImage( )에서 Mat 변환 함수를 호출한다.
  - 함수 `string type2str(int type)`이 추가되어 있다.

CDib to Mat 변환 함수

which 값에 따라 필요한 변수 포인터 배정하는 부분

```
void SWL01::DIBtoMat(int which) {
  CDib  *pDibF;
  Mat   *pMat;
  int   w, h, depth;
  if (which == 1) {
    pDibF = &m_dibFile1; pMat = &m_Mat1;
    w = m_width1; h = m_height1;
    depth = m_depth1;  // bits/pixcel
  }
  else if (which == 2) {
    pDibF = &m_dibFile2; pMat = &m_Mat2;
    w = m_width2; h = m_height2;
    depth = m_depth2;
  }
  (*pMat).release();  // clear any previous data
```

✦CDib to Mat 변환 함수 (계속)

```
  switch (depth) {
    case 8:
      *pMat = Mat(h, w, CV_8UC1); // allocate Mat
      break;
    case 16:  // we use CV_16UC1(not drawable)
      *pMat = Mat(h, w, CV_16UC1);
      break;
    case 24:
      *pMat = Mat(h, w, CV_8UC3);
  };
  unsigned char *pMatData = (*pMat).data;
  unsigned char *pDibData = (*pDibF).m_lpImage;
  //***이미지를 복사하는 코드를 가능한 짧게 작성하자***

  //***
  flip((*pMat), (*pMat), 0); // upsize down
}    // Mat은 CDib와 수직으로 방향이 반대다.
```

- PopUp drawing 함수 drawMatPopUp( )
  - 이 함수는 이해하기 어렵지 않을 것이다.
  - which 값에 따라 제목을 만든 후 imshow( )를 호출한다.
  - waitKey(0)에 의하여 기다리고 있다가 창을 닫으면 기다림을 멈춘다.
  - 제목을 만드는 과정에서 string 다루는 코드는 살펴보고 익히도록 하자.
  - 문의 사항이 있으면 담당 조교에게 질문한다.

메뉴 편집

- 앞에서 메뉴 편집하는 방법과 동일한 방법으로 PopUp으로 이미지1 보기 메뉴를 활성화 시키자.
- ID는 ID_PopUpIM01으로 하고 클래스는 C…Doc, 이벤트 처리 함수 이름은 OnPopUpIM01로 정한다.
- 이미 입력 이미지에 대한 Mat 자료를 구성하였기에 이벤트 처리기에서는 readImageF1이 true일 때 drawMatPopUp(1)만 호출하면 된다.
- 완성하면 빌드 후 시험해 보자.
- 프로그램 확인 후 PopUp으로 이미지2 보기 메뉴를 활성화 시키자.

  이때 ID는 ID_PopUpIM02, 클래스는 C…Doc, 이벤트 처리 함수 이름은 OnPopUpIM02로 정한다.

Mat m_MatR을 위한 LPBITMAPINFO 구조 생성

- 응용 문제를 실습하려면 처리 결과를 출력할 수 있는 함수가 필요하다.
  - 처리 이미지의 크기, 색 등 원본과 다를 수 있기 때문에 CDib 클래스의 이미지 출력 함수인 Draw( )를 사용할 수 없다.
  - 함수 Draw( )에서는 Microsoft가 제공하는 DIB 출력 함수인 StretchDIBits( )을 사용하는데, 이 함수는 bitmap header 구조인 LPBITMAPINFO를 요구한다.
  - LPBITMAPINFO 는 BITMAPINFOHEADER 구조체와 optional pallet(RGBQUAD)으로 구성되어 있다.
  - 따라서, Mat 이미지를 SDI 화면에 출력하기 위해서는 LPBITMAPINFO 구조를 직접 만들어야 한다.

- s074419L01_Mfc.cpp
  - 첨부한 Code11.txt의 LPBITMAPINFO 구조를 만드는 멤버 함수 Create_bmiHeader( )를 추가한다.
  - 내용을 s074419L01_Mfc.cpp에 추가한다.
- s074419L01.h
  - 위 함수를 SWL01의 멤버 함수로 등록.
  - 인포헤더를 저장할 장소를 private 영역에 할당.

```
private:
BYTE tmp[sizeof(BITMAPINFO) + 255*sizeof(RGBQUAD)];
```

  - 이미지 처리 결과가 저장될 Mat m_MatR에 대한 인포헤더 mg_lpBMIHR 선언.

```
LPBITMAPINFO mg_lpBMIHR = (LPBITMAPINFO)&tmp;
```

  - 이러한 수정을 적용한 Code12.txt를 s074419L01.h에 복사한다.

◆ LPBITMAPINFO 생성 함수

✦BITMAPINFOHEADER 구조 생성

```
void SWL01::Create_bmiInfoHeader(cv::Mat *image) {
  int bpp = image->channels() * 8;
  int w = image->cols, h = image->rows;
  memset(mg_lpBMIHR, 0, sizeof(BITMAPINFO));
  mg_lpBMIHR->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
  mg_lpBMIHR->bmiHeader.biPlanes = 1;
  mg_lpBMIHR->bmiHeader.biBitCount = bpp;
  mg_lpBMIHR->bmiHeader.biCompression = BI_RGB;
  mg_lpBMIHR->bmiHeader.biWidth = w;
  mg_lpBMIHR->bmiHeader.biHeight = -h; // 거꾸로 출력
  mg_lpBMIHR->bmiHeader.biSizeImage = w * h * 1;
```

✦Pallet (RGBQUAD) 구조 생성

```
  switch (bpp) {
  case 8:
    for (int i = 0; i < 256; i++) {
      mg_lpBMIHR->bmiColors[i].rgbBlue = (BYTE)i;
      mg_lpBMIHR->bmiColors[i].rgbGreen = (BYTE)i;
      mg_lpBMIHR->bmiColors[i].rgbRed = (BYTE)i;
      mg_lpBMIHR->bmiColors[i].rgbReserved = 0;
    }
    break;
  case 32:
  case 24:
    ((DWORD*)mg_lpBMIHR->bmiColors)[0] = 0x00FF0000; // red
    ((DWORD*)mg_lpBMIHR->bmiColors)[1] = 0x0000FF00; // green
    ((DWORD*)mg_lpBMIHR->bmiColors)[2] = 0x000000FF; // blue
    break;
  }
}
```

◆ 함수 DrawImage( ) 수정

✦ which == 3일 때 m_MatR의 이미지를 출력하도록 Code13.txt 와 같이 수정한다.

✦ 실제로 추가한 부분은 다음과 같다:

```
else if (which == 3) {
  StretchDIBits( pDC->GetSafeHdc(),        // dc handle
    dcLTx, dcLTy,                          // 화면의 좌상귀 좌표
    m_MatR.cols, m_MatR.rows,      // 화면의 폭과 높이
    0, 0,                                  // Mat 배열의 좌상귀
    m_MatR.cols, m_MatR.rows,      // Mat 배열의 폭과 높이
    m_MatR.data,                           // image data to display
    mg_lpBMIHR,                  // BITMAPINFO 시작 주소
    DIB_RGB_COLORS,              // RGB 또는 색상 테이블 인덱스
    SRCCOPY                      // 래스터 연산 방법
    );
}
```

OnDraw( ) 수정

- which == 3일 때는 processedF가 true일 경우 이미지 2 대신 처리 결과인 m_MatR을 출력하도록 한다.
- 따라서, 이미지 2는 이미 읽었더라도 processedF가 false일 때만 출력한다.
- 아래 이 함수의 코드를 보인다.

```
if (SWL01_inst.readImageF1 == true) {
  SWL01_inst.drawImage(pDC, 20, 20, 1);
}
if (SWL01_inst.readImageF2 == true &&
      SWL01_inst.processedF == false) {
  SWL01_inst.drawImage(pDC, 20, 20, 2);
}
if (SWL01_inst.processedF == true) {
  SWL01_inst.drawImage(pDC, 20, 20, 3);
}
```

- OnDraw( ) 함수는 Code14.txt의 내용으로 교체한다.

간단한 이미지 처리 문제 프로그래밍

- 이제 응용 문제를 프로그래밍해 보자.
- 이제부터 응용 문제에 대한 함수는 snnnnnnL01_App.cpp에 작성하도록 한다.
  - 단, 처리 후 결과를 보는 프로그램은 snnnnnnL01_Mfc.cpp에 작성한다.
- 실습을 마친 후 이 두 파일은 주어진 기간 내에 사이버 캠퍼스의 과제 제출 항에 제출하여야 한다.
- 이 두 프로그램을 지금까지 만든 **MFC** 프레임에 추가하여 테스트 할 것이므로 이벤트 처리 함수 이름, 함수가 속한 클래스 등을 안내한 이름으로 정확히 작성하여야 한다.

응용 1 : 이미지를 어둡게 하기

- 8 bit grayscale 이미지를 읽어 모든 pixel 값을 반으로 줄여보자.
  - 이미지 1(m_Mat1)에 대해 이 기능을 수행하도록 만든다.
  - 1주차 메뉴 어둡게 하기를 활성화 시키고 이 메뉴에 대해 처리가 가능하도록 프로그래밍한다.
  - 메뉴 ID는 ID_DarkenImg로 하고 이벤트 처리기의 소속 클래스와 이름은 각각 CswLab19fApp, OnDarkenImage로 정하자.
  - 이를 처리하기 위한 클래스 SWL01의 멤버 함수 이름은

    void DarkenGrayscaleImage(void)로 한다.
  - OnDarkenImage( )에서 DarkenGrayscaleImage( ) 호출은 반드시 SWL01 instance SWL01_inst를 통해야 함을 잊지 말자.
  - 이미지를 어둡게 하는 것은 pixel 값을 일률적으로 작게 하면 되는데, 여기서는 본래 Pixel 값을 반으로 줄여 m_MatR에 저장하는 것으로 한다.

- 함수 void DarkenGrayscaleImage(void)
  - 이 함수는 s074419L01_App.cpp에 저장해야 하며, 이 함수에서 해야할 일은 다음과 같다.
  - 이미지 1을 읽었는지 체크하고(readImageF1), 아니면 오류 메시지를 출력하고 return한다. 오류 메시지는 함수 AfxMessageBox( )를 사용한다.
    - AfxMessageBox( )의 사용 예는 snnnnnnL01_Mfc.cpp의 멤버 함수 readImage( )에서 찾을 수 있다.
  - 읽은 이미지가 8 bit grayscale 이미지인지 확인하고 아니면 오류 메시지를 출력하고 끝낸다.
    - m_Mat1의 타입이 CV_8UC1인지 체크하면 된다.
    - snnnnnnL01_Mfc.cpp의 함수 drawMatPopUp( )에 이를 체크하는 예가 있으니 참고하자.
    - 이 역시 조건에 맞지 않으면 AfxMessageBox( ) 를 사용하여 오류 메시지를 보이고 바로 return 한다.

- 입력 오류 체크 후 이상이 없으면
  - m_MatR을 먼저 release하고, 크기가 m_Mat1과 같은 CV_8UC1 type의 배열을 다시 m_MatR에 생성한다.
  - 원소 크기를 반으로 줄이는 것은 앞에서 설명한 Mat 데이터 읽는 예를 통하여 쉽게 구현할 수 있을 것이다.
  - 혹은 Mat 연산을 사용하여 간단히 해결할 수도 있다(1).
  - 연산 후 함수 마지막에 다음 코드를 추가 하여야 한다.

```
Create_bmiInfoHeader(&m_MatR);  // 인포헤더를 갱신
processedF = true; // 처리 완료를 flag을 통하여 알린다
g_pView->Invalidate();   // OnDraw를 호출한다
```

- 또한, s074419L01.h에 DarkenGrayscaleImage( )을 멤버 함수로 등록해야 한다.

(1) https://docs.opencv.org/2.4.13.6/modules/core/doc/operations_on_arrays.html

- 함수 작성을 마친 후
  - 빌드 후 실행하여 보면 결과가 화면 우측에 보일 것이다.
  - 추가로 PopUp으로 결과 보기를 활성화시켜 PopUp 창으로도 출력을 볼 수 있게 기능을 추가시키자(간단함).
  - 이를 위해 ID는 ID_PopUpIMR로 하고 이벤트 처리기의 클래스와 함수 이름을 각각 CswLab19fDoc와 OnPopUpIMR로 정한다.
  - OnPopUpIMR 에서는 processedF = true일 경우, SWL01_inst.drawMatPopUp(3)으로 drawing 함수를 호출한다.
- 빌드 후 프로그램을 체크해 본다.

결과 이미지 파일에 쓰기

- 이제 MFC 함수 작성 마지막으로 이미지 처리 결과를 파일에 쓰는 기능을 추가하자.
  - 메뉴 편집기에서 처리 결과 저장의 ID를 ID_SAVE_IMG, 클래스와 이벤트 처리기를 각각 CswLab19fDoc 와 OnSaveImage로 설정한다.
  - 이벤트 처리기 OnSaveImage( )에는 Code15.txt의 내용을 복사한다.
  - 이 함수 역시
    - 파일 선택 창이 뜨고, 거기서 파일 이름을 입력하거나 overwrite 할 파일을 선택하게 하고,
    - SWL01의 멤버 함수 saveImage(pathName)을 호출하게 되어 있다.

◆ 함수 saveImage(pathName)

✦ Code16.txt의 내용을 s074419L01_Mfc.cpp에 복사하고, s074419L01.h에 이 함수를 멤버 함수로 등록하자.

✦ 이미지 저장은 imwrite( )를 사용하는데, 이 함수는 주어진 파일 이름의 extension을 보고 저장 형식을 결정한다.

```
void SWL01::saveImage(CString pathName) {
  string str = CT2CA(pathName);
  const char *cstr = str.c_str();
  bool result = imwrite(cstr, m_MatR);
  if (!result) {
    AfxMessageBox(L"ERROR: Cannot Save The Image",
                  MB_OK, 0);
  }
  g_pView->Invalidate(TRUE);
  return;
}
```

✦ 이미지를 어둡게 한 후 이를 저장해서 동작을 확인하자.

응용 2 : 이미지를 밝게 하기

- 8 bit grayscale 이미지를 읽어 모든 pixel 값을 두 배로 하자.
  - 응용 1과 거의 같은 순서로 프로그래밍 한다.
  - 1주차 메뉴 밝게 하기를 활성화 시키고 이 메뉴에 대해 처리가 가능하도록 프로그래밍한다.
  - 메뉴 ID는 ID_BrightenImg로 하고 이벤트 처리기의 소속 클래스와 이름은 각각 Csw..App, OnBrightenImage로 정하자.
  - 이를 처리하기 위한 클래스 SWL01의 멤버 함수 이름은

    void BrightenGrayscaleImage(void)로 한다.
  - 이미지를 밝게 하는 것은 pixel 값을 일률적으로 크게 하면 되는데, 여기서는 본래 Pixel 값을 두 배로 늘려 m_MatR에 저장하는 것으로 한다.

- 함수 void BrightenGrayscaleImage(void)
  - 이 함수의 구성은 함수 DarkenGrayscaleImage과 행렬 연산을 제외하고는 동일하며 s074419L01_App.cpp에 저장하여야 한다.
  - 그런데 각 pixel 값을 단순히 두 배로 하여 영상을 보면, 아주 이상하게 보일 것이다.
    - 이 이유를 생각해 보고, 정상적으로 밝게 보이도록 프로그램을 작성해보자.

응용 3 : 24 bits Color 이미지를 Grayscale로 변환

- 24 bits Color 이미지를 grayscale 이미지로 변환하자.
  - 이 역시 응용 1과 거의 같은 순서로 프로그래밍 한다.
  - 1주차 메뉴 Color24를 Grayscale로를 활성화 시키고 이 메뉴에 대해 처리가 가능하도록 프로그래밍한다.
  - 메뉴 ID는 ID_24ColToGray로 하고 이벤트 처리기의 소속 클래스와 이름은 각각 Csw…App, On24ColorToGray로 정하자.
  - 이를 처리하기 위한 클래스 SWL01의 멤버 함수 이름은 void Color24toGrayscale(void)로 한다.

- 함수 void BrightenGrayscaleImage(void)
  - 이 함수의 구성은 함수 DarkenGrayscaleImage과 거의 동일하며 s074419L01_App.cpp에 저장하여야 한다.
  - 입력 이미지의 type은 CV_8UC3이어야 한다. 아니면 오류 출력.
  - Grayscale 이미지를 만드는 것이므로 m_MatR의 type은 CV_8UC1으로 생성해야 한다.
  - Color 이미지를 grayscale로 변환하는 것은 NTSC 방식인 다음 식에 의한다:

$$GRAY = 0.299R + 0.587G + 0.114B$$

  - 이 변환을 수행하는 OpenCV 함수를 인터넷에서 찾아보고 보다 간단한 코드를 작성해도 좋다.

응용 4 : 16 bits Color 이미지를 Grayscale로 변환

- 16 bits Color 이미지를 grayscale 이미지로 변환하자.
  - 이 역시 응용 1과 거의 같은 순서로 프로그래밍 한다.
  - 1주차 메뉴 Color16를 Grayscale로를 활성화 시키고 이 메뉴에 대해 처리가 가능하도록 프로그래밍한다.
  - 메뉴 ID는 ID_16ColToGray로 하고 이벤트 처리기의 소속 클래스와 이름은 각각 Csw...App, On16ColToGray로 정하자.
  - 이를 처리하기 위한 클래스 SWL01의 멤버 함수 이름은 void Color16toGrayscale(void)로 한다.

- 함수 void BrightenGrayscaleImage(void)
  - 이 함수의 구성 역시 함수 DarkenGrayscaleImage과 거의 동일하며 s074419L01_App.cpp에 저장하여야 한다.
  - 입력 이미지의 type은 CV_16UC1이어야 한다. 앞에서 이 type으로 저장했기 때문이면 아니면 오류 출력한다.
    - 따라서, 입력 이미지의 pixel 값은 unsigned short으로 처리한다.
  - Grayscale 이미지를 만드는 것이므로 m_MatR의 type은 CV_8UC1으로 생성해야 한다.
  - Color 이미지를 grayscale로 변환하는 것은 NTSC 방식인 다음 식에 의한다:

$$GRAY = 0.299R + 0.587G + 0.114B$$

  - 다음 쪽의 주의 사항을 읽어 보고 프로그래밍하자.

- 주의 사항
  - 16bit Color 이미지는 5 bits 씩 B(LSD), G, R로 저장되어 있으므로 shift와 bitwise and 연산을 사용하여, 값을 얻은 후, 앞에 보인 NTSC 수식을 적용해야 한다.
  - 그런데 5 bit 데이터 값은 최대 값이 31이므로 수식에 적용하여 그 결과를 그대로 적용하면 이미지가 부자연스럽게 보일 것이다.
  - 따라서 변환 결과를 그 최대 값이 255가 되도록 scale up해야 한다.