BMP 파일 처리

김성영교수
금오공과대학교
컴퓨터공학과

학습 내용

- 영상 반전 프로그램 제작

Inverting images

out = 255 − in

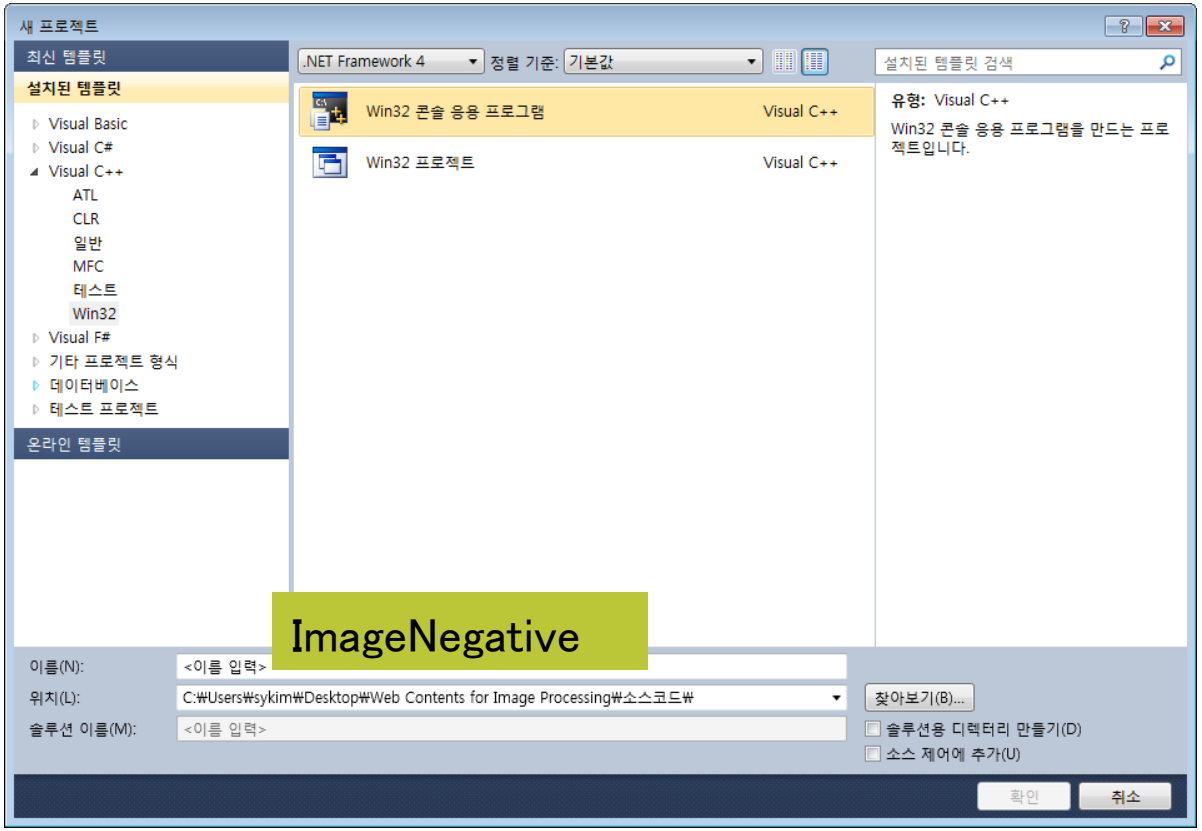
```c
/*
   이 프로그램은 8bit gray-scale 영상을 입력으로 사용하여 반전한 후
   동일 포맷의 영상으로 저장한다.
*/
#include <stdio.h>
#include <windows.h>

#define WIDTHBYTES(bytes) (((bytes)+3)/4*4)

int main( int argc, char *argv[] )
{
    FILE *file;             // file pointer
    BITMAPFILEHEADER hf;    // 파일헤더 (bmp file header)
    BITMAPINFOHEADER hInfo;   // 비트맵 정보헤더 (bitmap information header)
    RGBQUAD rgb[256];       // Lookup Table
    int widthStep;          // 라인 당 바이트 (bytes per a line)
    BYTE *lpImg;            // 입력 데이터 포인터 (pointer for input image data)
    BYTE *lpOutImg;         // 출력 데이터 포인터 (pointer for output image data)
    int x, y;

    if( argc < 3 ) {
        printf( "Insufficient Input Arguments \n" );
        printf( "  invertImage input_file ouput_file \n" );
        return -1;
    }
```

```
// 입력 영상 파일을 연다
file = fopen( argv[1], "rb" );
if( file == NULL ) {
    printf( "이미지 파일을 열 수 없습니다! \n" );
    return -1;
}

fread( &hf, sizeof(BITMAPFILEHEADER), 1, file ); // 파일 헤더 읽음
if( hf.bfType != 0x4D42 ) { // BMP 포맷 ('BM') 인지를 확인
    printf( "BMP 파일이 아닙니다. \n" );
    return -1;
}

fread( &hInfo, sizeof(BITMAPINFOHEADER), 1, file ); // 비트맵 정보헤더 읽음

printf( "Size: (%3dx%3d) \n", hInfo.biWidth, hInfo.biHeight ); // 크기정보 출력

// 8bit gray-scale 영상을 확인
if( hInfo.biBitCount != 8 || hInfo.biClrUsed != 0 ) {
    printf( "8bit gray-scale 영상이 아닙니다..!! \n" );
    return -1;
}

// Lookup Table 읽음
fread( rgb, sizeof(RGBQUAD), 256, file ); // Lookup Table 읽음
```

```
// 입력 데이터를 위한 라인당 바이트 수 계산
widthStep = WIDTHBYTES( (hInfo.biBitCount/8) * hInfo.biWidth );

fseek( file, hf.bfOffBits, SEEK_SET );   // 비트맵 데이터가 시작하는 위치로 이동

// 입력 데이터를 저장하기 위한 메모리 할당
lpImg = (BYTE *)malloc( widthStep * hInfo.biHeight );

// 입력영상에서 영상 데이터를 읽음
fread( lpImg, sizeof(BYTE), widthStep*hInfo.biHeight, file );

fclose( file );

// 결과 데이터를 저장하기 위한 메모리 할당
lpOutImg = (BYTE *)malloc( widthStep * hInfo.biHeight );

// 영상 반전 연산
for( y=0; y<hInfo.biHeight; y++ ) {
    for( x=0; x<hInfo.biWidth; x++ ) {
        lpOutImg[y*widthStep + x] = 255 - lpImg[y*widthStep + x];
    }
}
```

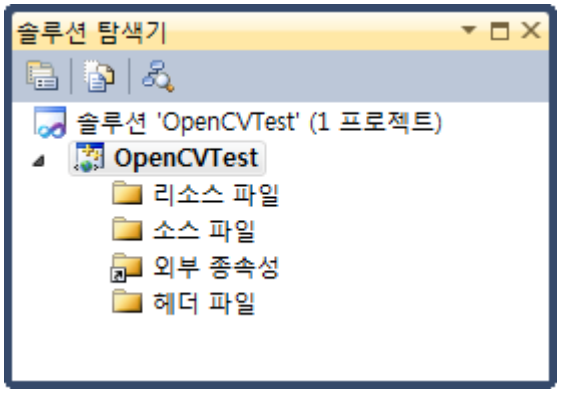
```
    file = fopen( argv[2], "wb" );

    fwrite( &hf, sizeof(char), sizeof(BITMAPFILEHEADER), file );
    fwrite( &hInfo, sizeof(char), sizeof(BITMAPINFOHEADER), file );
    fwrite( rgb, sizeof(RGBQUAD), 256, file );

    fseek( file, hf.bfOffBits, SEEK_SET );   // 비트맵 데이터가 시작하는 위치로 이동

    fwrite( lpOutImg, sizeof(BYTE), widthStep*hInfo.biHeight, file );

    fclose( file );

    // 메모리 해제|
    free( lpOutImg );
    free( lpImg );

    return 0;
}
```

파일|새로 만들기|프로젝트… File|New Project|Project…

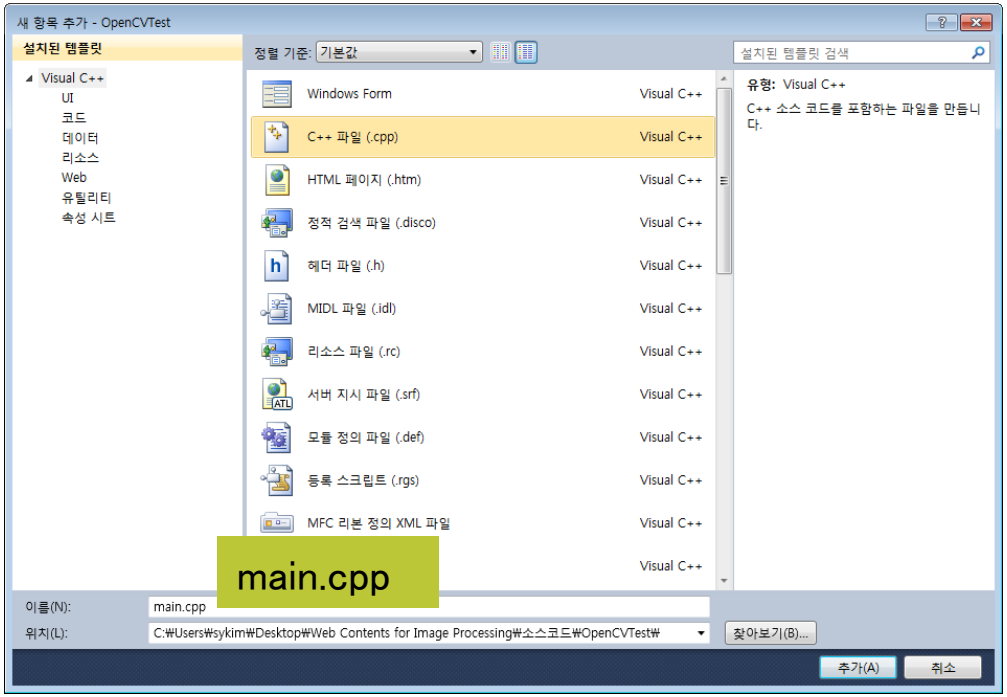


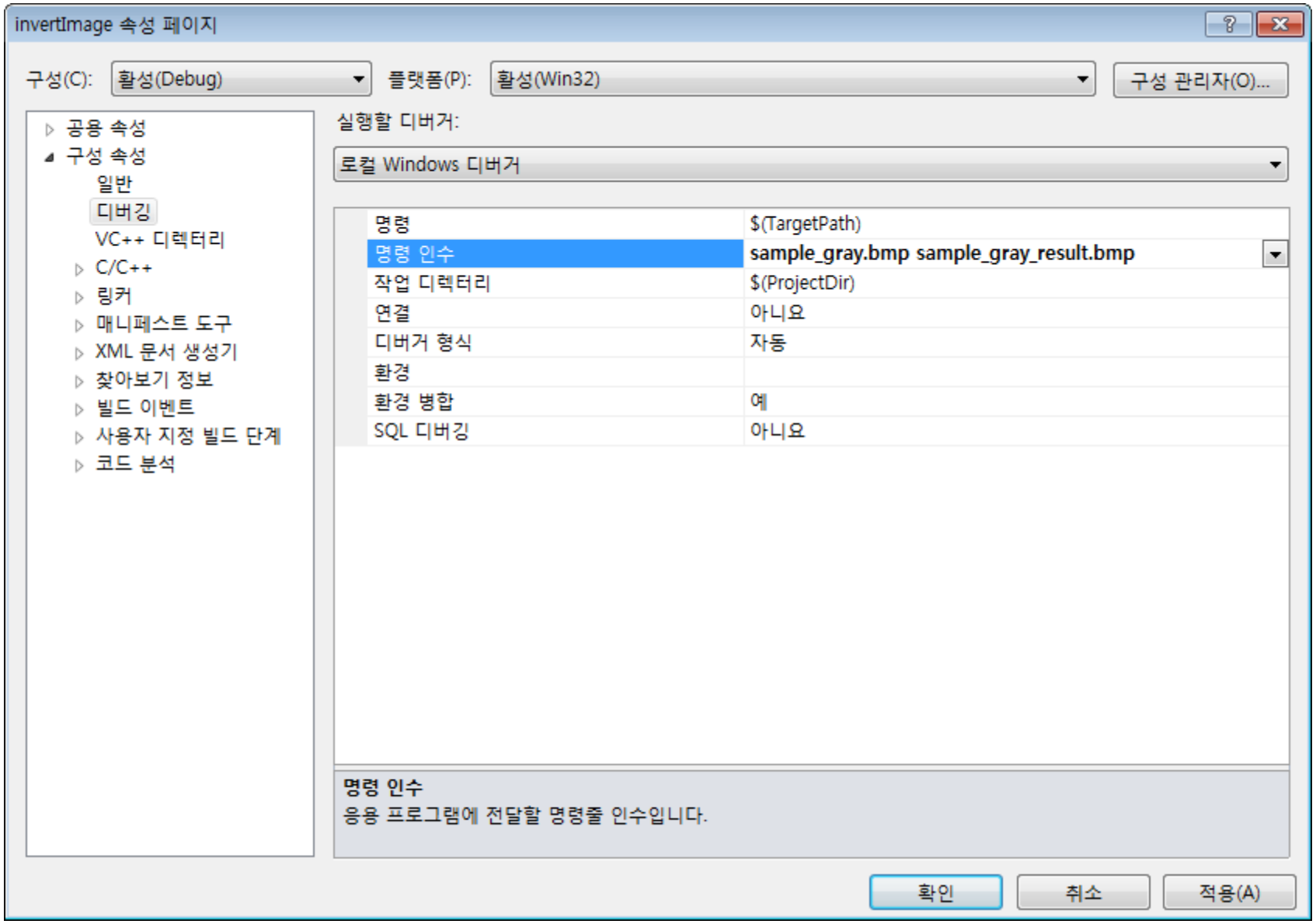
콘솔 응용 프로그램 | 빈 프로젝트 Console Application | Empty Project

보기| 솔루션 탐색기 View|Solution Explorer

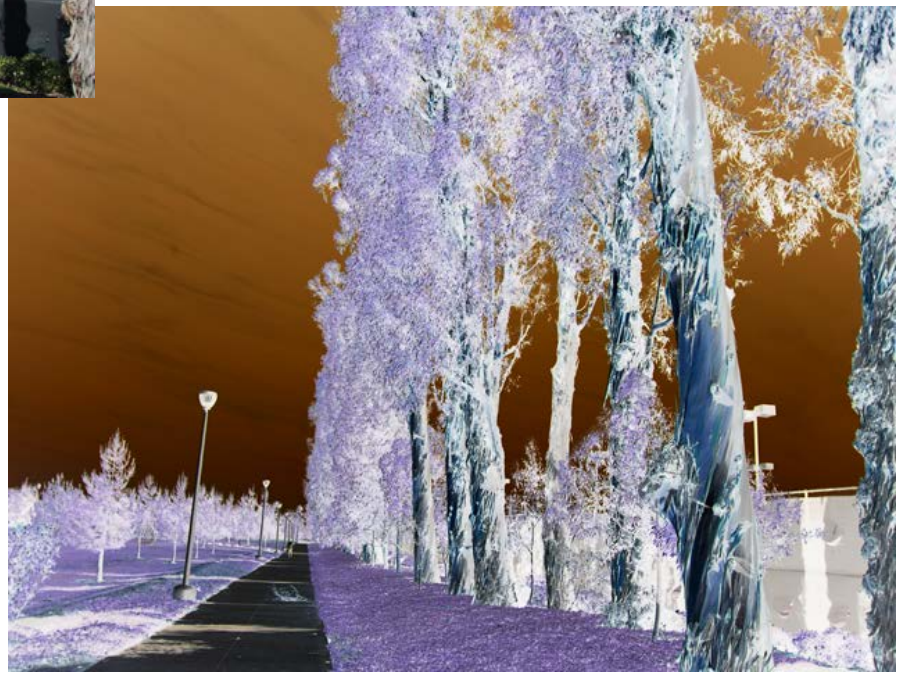


1. Right-clicking on **소스파일** Source Files
2. **추가|새 항목...** Add New Item...




invertImage 속성 페이지
구성(C): 활성(Debug)
플랫폼(P): 활성(Win32)
구성 관리자(O)...
공용 속성
구성 속성
일반
디버깅
VC++ 디렉터리
C/C++
링커
매니페스트 도구
XML 문서 생성기
찾아보기 정보
빌드 이벤트
사용자 지정 빌드 단계
코드 분석
실행할 디버거:
로컬 Windows 디버거
명령 $(TargetPath)
명령 인수 sample_gray.bmp sample_gray_result.bmp
작업 디렉터리 $(ProjectDir)
연결 아니요
디버거 형식 자동
환경
환경 병합 예
SQL 디버깅 아니요
명령 인수
응용 프로그램에 전달할 명령줄 인수입니다.
확인
취소
적용(A)

True color images에 대해서도 사용가능하도록 소스코드를 변경하자!



```
/*
   이 프로그램은 8bit gray-scale 및 true color 영상을 입력으로 사용하여 반전한 후
   동일 포맷의 영상으로 저장한다.
*/
#include <stdio.h>
#include <windows.h>

#define WIDTHBYTES(bytes) (((bytes)+3)/4*4)

int main( int argc, char *argv[] )
{
    FILE *file;             // file pointer
    BITMAPFILEHEADER hf;    // 파일헤더 (bmp file header)
    BITMAPINFOHEADER hInfo;   // 비트맵 정보헤더 (bitmap information header)
    RGBQUAD rgb[256];       // Lookup Table
    int widthStep;          // 라인 당 바이트 (bytes per a line)
    BYTE *lpImg;            // 입력 데이터 포인터 (pointer for input image data)
    BYTE *lpOutImg;         // 출력 데이터 포인터 (pointer for output image data)
    int x, y;

    if( argc < 3 ) {
        printf( "Insufficient Input Arguments \n" );
        printf( "  invertImage input_file ouput_file \n" );
        return -1;
    }
```

```
// 입력 영상 파일을 연다
file = fopen( argv[1], "rb" );
if( file == NULL ) {
    printf( "이미지 파일을 열 수 없습니다! \n" );
    return -1;
}

fread( &hf, sizeof(BITMAPFILEHEADER), 1, file ); // 파일 헤더 읽음
if( hf.bfType != 0x4D42 ) { // BMP 포맷 ('BM') 인지를 확인
    printf( "BMP 파일이 아닙니다. \n" );
    return -1;
}

fread( &hInfo, sizeof(BITMAPINFOHEADER), 1, file ); // 비트맵 정보헤더 읽음

printf( "Size: (%3dx%3d) \n", hInfo.biWidth, hInfo.biHeight ); // 크기정보 출력

// 8bit gray-scale 및 true color 영상을 확인
if( (hInfo.biBitCount!=8 || hInfo.biClrUsed!=0) && hInfo.biBitCount!=24 ) {
    printf( "8bit gray-scale 영상이 아닙니다..!! \n" );
    return -1;
}

// Lookup Table 읽음
if( hInfo.biBitCount == 8 ) {
    fread( rgb, sizeof(RGBQUAD), 256, file ); // Lookup Table 읽음
}
```

```
// 입력 데이터를 위한 라인당 바이트 수 계산
widthStep = WIDTHBYTES( (hInfo.biBitCount/8) * hInfo.biWidth );

fseek( file, hf.bfOffBits, SEEK_SET );   // 비트맵 데이터가 시작하는 위치로 이동

// 입력 데이터를 저장하기 위한 메모리 할당
lpImg = (BYTE *)malloc( widthStep * hInfo.biHeight );

// 입력영상에서 영상 데이터를 읽음
fread( lpImg, sizeof(BYTE), widthStep*hInfo.biHeight, file );

fclose( file );

// 결과 데이터를 저장하기 위한 메모리 할당
lpOutImg = (BYTE *)malloc( widthStep * hInfo.biHeight );
```

```
// 영상 반전 연산
if( hInfo.biBitCount == 24 ) {
    for( y=0; y<hInfo.biHeight; y++ ) {
        for( x=0; x<hInfo.biWidth; x++ ) {
            lpOutImg[y*widthStep+3*x+2] = 255-lpImg[y*widthStep+3*x+2];  /* R */
            lpOutImg[y*widthStep+3*x+1] = 255-lpImg[y*widthStep+3*x+1];  /* G */
            lpOutImg[y*widthStep+3*x+0] = 255-lpImg[y*widthStep+3*x+0];  /* B */
        }
    }
}
else if( hInfo.biBitCount == 8 ) {
    for( y=0; y<hInfo.biHeight; y++ ) {
        for( x=0; x<hInfo.biWidth; x++ ) {
            lpOutImg[y*widthStep + x] = 255 - lpImg[y*widthStep + x];
        }
    }
}
```

```
    file = fopen( argv[2], "wb" );

    fwrite( &hf, sizeof(char), sizeof(BITMAPFILEHEADER), file );
    fwrite( &hInfo, sizeof(char), sizeof(BITMAPINFOHEADER), file );
    if( hInfo.biBitCount == 8 ) {
        fwrite( rgb, sizeof(RGBQUAD), 256, file );
    }

    fseek( file, hf.bfOffBits, SEEK_SET );   // 비트맵 데이터가 시작하는 위치로 이동

    fwrite( lpOutImg, sizeof(BYTE), widthStep*hInfo.biHeight, file );

    fclose( file );

    // 메모리 해제|
    free( lpOutImg );
    free( lpImg );

    return 0;
}
```