

8장 파일 입출력

2022년도 1학기 윈도우 프로그래밍

학습 목표

- 학습목표
 - API에서 제공하는 파일 입출력을 배운다.
 - 공용 대화상자 사용법에 대하여 배운다.
- 내용
 - 파일 다루기
 - 공용 대화상자

1. 파일 다루기

- API 이용한 표준 입출력 및 파일 사용 방법
 - 파일을 만들고 열어준다. 열 때는 읽기용인지 쓰기용인지 명시한다.
 - 열린 파일에는 텍스트를 쓰거나 읽는다.
 - 작업 후에는 파일을 닫는다.

기능	C언어 표준 라이브러리 함수	Win32 API함수
파일 열기	fopen()	CreateFile()
파일 닫기	fclose()	CloseHandle()
파일 포인터 위치변경/획득	fseek()	SetFilePointer()
파일 읽기	fread()	ReadFile()
파일 쓰기	fwrite()	WriteFile()

파일 생성/열기 함수

- 파일 생성 함수

HANDLE CreateFile (LPCTSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDisposition, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile);

- 파일을 생성하거나 열 수 있다.
 - **lpFileName**: 만들 파일 이름
 - **dwDesiredAccess**: 읽기/쓰기 모드 (아래의 3 모드 중 1개 지정)
 - 읽기: **GENERIC_READ**
 - 쓰기: **GENERIC_WRITE**
 - 읽기 및 쓰기: **GENERIC_READ | GENERIC_WRITE**
 - **dwShareMode**: 공유 모드 (파일 공유 여부 명시)
 - 읽기 공유허용: **FILE_SHARE_READ**
 - 쓰기 공유허용: **FILE_SHARE_WRITE**
 - **lpSecurityAttributes**: 보안 속성 (자녀 프로세스에 상속 여부 설정), NULL 이면 상속 안됨
 - **dwCreationDisposition**: 파일 생성 모드
 - 새로 만들기, 이미 있으면 에러 메시지: **CREATE_NEW**
 - 항상 새로 만들기, 파일이 있어도 파괴하고 새로 만들: **CREATE_ALWAYS**
 - 기존 파일 열기, 파일이 없으면 에러 메시지: **OPEN_EXISTING**
 - 항상 열기: **OPEN_ALWAYS**
 - **dwFlagsAndAttributes**: 파일 속성 (읽기 전용 파일, 시스템 파일, 숨겨진 파일 등 지정)
 - 일반적인 파일: **FILE_ATTRIBUTE_NORMAL**
 - **hTemplateFile**: 생성될 파일의 속성을 제공할 템플릿 파일

파일 생성/열기 함수

- 파일 읽기

```
BOOL ReadFile (HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead,  
               LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);
```

- 파일 읽기 함수
 - hFile: 데이터를 읽을 파일 핸들
 - lpBuffer: 읽은 자료를 넣을 버퍼
 - nNumberOfBytesToRead: 읽고자 하는 바이트 크기
 - lpNumberOfBytesRead: 실제 읽은 바이트
 - lpOverlapped: NULL

- 파일 쓰기

```
BOOL WriteFile (HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToWrite,  
               LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);
```

- 파일 쓰기 함수
 - hFile: 데이터를 저장할 파일 핸들
 - lpBuffer: 쓸 자료가 저장된 버퍼
 - nNumberOfBytesToWrite : 쓸 자료의 바이트 크기
 - lpNumberOfBytesWritten : 실제 쓴 바이트
 - lpOverlapped: NULL

- 파일 닫기

```
void CloseFile (HANDLE hFile);
```

- 파일 닫기 함수
 - hFile: 닫을 파일 핸들

임의 접근

- 파일 액세스할 때 대상 파일 위치 (File Pointer) 결정
 - 최초로 파일이 열렸을 때: FP는 파일의 선두 위치, 파일을 읽거나 쓰면 그만큼 파일 포인터가 이동 → 순차적 접근
 - 파일의 임의의 위치에서 원하는 만큼 읽는다. → 임의 접근

```
DWORD SetFilePointer (HANDLE hFile, LONG lDistanceToMove,  
                     PLONG lpDistanceToMoveHigh, DWORD dwMoveMethod);
```

- 임의 접근 함수
 - hFile: 파일 핸들
 - lDistanceToMove: 파일 포인터가 이동할 위치
 - lpDistanceToMoveHigh: 파일 크기가 2GB 이상일 경우 옮길 위치
 - dwMoveMethod: 파일 포인터의 이동 시작 위치 지정
 - FILE_BEGIN / FILE_CURRENT / FILE_END

파일 입출력 예

- 사용 예) 기존 텍스트파일을 열어 화면에 출력하고, 데이터를 파일에 저장하기

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

```
{
    HDC hdc;
    HANDLE hFile;
    TCHAR InBuff[1000];
    TCHAR OutBuff[1000] = L"API 파일 입출력 테스트입니다.";
    int size = 1000, read_size;

    switch (iMsg) {
    case WM_LBUTTONDOWN:
        hFile = CreateFile ( L"test.txt", GENERIC_READ|GENERIC_WRITE, FILE_SHARE_READ|FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, 0);
        memset (InBuf, 0, sizeof (InBuff));
        ReadFile (hFile, InBuff, size, &read_size, NULL);           //--- hFile에서 size 만큼 읽어 InBuff에 저장
        InBuff[read_size] = '\0';

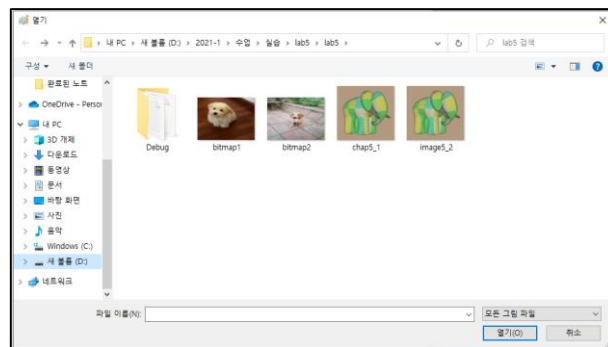
        hdc = GetDC(hwnd);
        TextOut (hdc, 0, 0, InBuff, lstrlen(InBuff));              //--- InBuff 에 있는 내용을 화면에 출력
        ReleaseDC (hwnd, hdc);

        SetFilePointer (hFile, 0, NULL, FILE_END);
        WriteFile (hFile, OutBuff, lstrlen(OutBuff)*sizeof(TCHAR), &size, NULL);    //--- OutBuff의 내용을 hFile의 끝에 저장
        CloseHandle (hFile);

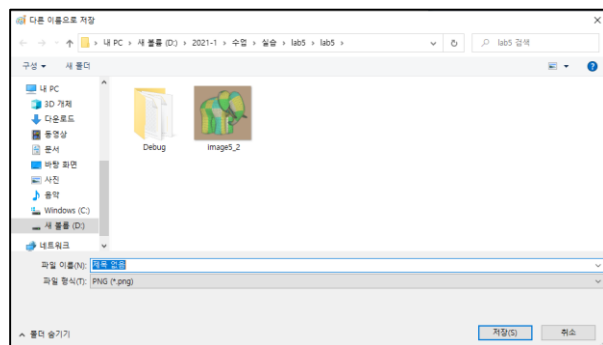
        break;
    }
```

2. 공용대화상자

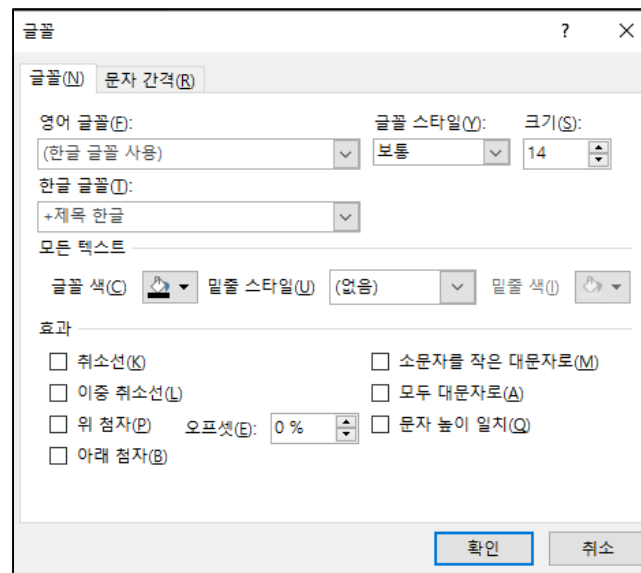
- 윈도우의 공용 대화상자
 - 파일 열기, 글꼴 선택하기, 색상 선택하기 등의 윈도우
 - 해당 대화상자의 구조체 설정 ➔ 함수 호출



파일 열기



파일 저장하기



글꼴 선택하기

1) 공용대화상자: 파일 열기

- 파일열기 처리절차
 - OPENFILENAME** 구조체 할당
 - 열기함수 호출

OPENFILENAME 구조체

```
typedef struct tagOFN{  
    DWORD   IStructSize;  
    HWND     hwndOwner;  
    HINSTANCE hInstance;  
    LPCTSTR  lpstrFilter;  
    LPTSTR   lpstrCustomFilter;  
    DWORD    nMaxCustFilter;  
    DWORD    nFilterIndex;  
    LPTSTR   lpstrFile;  
  
    DWORD    nMaxFile;  
    LPTSTR   lpstrFileTitle;  
    DWORD    nMaxFileTitle;  
    LPCTSTR  lpstrInitialDir;  
    LPCTSTR  lpstrTitle;  
    DWORD    Flags;  
    WORD     nFileOffset;  
    WORD     nFileExtension;  
    LPCTSTR  lpstrDefExt;  
    DWORD    lCustData;  
    LPOFNHOOKPROC lpfnHook;  
    LPCTSTR  lpTemplateName;  
}  
OPENFILENAME;
```

//--- ofn
//--- 구조체 크기
//--- 오너 윈도우 핸들
//--- 인스턴스 핸들
//--- 파일 형식 콤보 박스에 나타낼 필터
//--- 커스텀 필터를 저장하기 위한 버퍼
//--- 커스텀 필터 버퍼의 길이
//--- 파일 형식 콤보 박스에서 사용할 필터의 인덱스
//--- 파일 이름 에디트에 처음 나타낼 파일명
//--- 최종적으로 선택된 파일이름이 저장된다.
//--- lpstrFile 멤버의 길이
//--- 선택한 파일명을 리턴받기 위한 버퍼 (경로X)
//--- lpstrFileTitle 멤버의 길이
//--- 파일 찾기를 시작할 디렉토리
//--- 대화상자의 캡션
//--- 대화상자의 모양과 동작을 지정하는 플래그
//--- lpstrFile 버퍼 내의 파일명 오프셋
//--- lpstrFile 버퍼 내의 파일 확장자 오프셋
//--- 디폴트 확장자
//--- 혹 프로시저로 보낼 사용자 정의 데이터
//--- 혹 프로시저명
//--- 템플리트명

파일 공용 대화상자

- 파일 공용 대화상자를 열기 위한 함수

BOOL **GetOpenFileName** (LPOPENFILENAME lpofn);

- 파일 입출력을 위해 파일 공용 대화상자를 열어 대상 파일을 입력받기 위해 호출되는 함수
- lpofn: **입력**을 위한 OPENFILENAME 구조체 포인터

BOOL **GetSaveFileName** (LPOPENFILENAME lpshn);

- 파일 입출력을 위해 파일 공용 대화상자를 열어 대상 파일을 입력받기 위해 호출되는 함수
- lpshn: **출력**을 위한 OPENFILENAME 구조체 포인터

1) 공용대화상자: 파일 열기

- 파일열기 처리절차
 - **OPENFILENAME** 구조체 할당
 - 열기함수 호출 → 파일이름 획득

OPENFILENAME OFN;

//--- 구조체 선언

memset (&OFN, 0, sizeof(OPENFILENAME));

//--- 구조체 초기화

OFN.lStructSize = sizeof(OPENFILENAME);

OFN.hwndOwner = hwnd;

OFN.lpstrFilter = filter;

OFN.lpstrFile = lpstrFile;

OFN.nMaxFile = 256;

OFN.lpstrInitialDir = " . " ;

if (**GetOpenFileName (&OFN)!=0**) { **//--- 파일 함수 호출**

 wsprintf (str, "%s 파일을 여시겠습니까 ?", OFN.lpstrFile);

 MessageBox (hwnd, str, L"저장하기 선택", MB_OK);

}

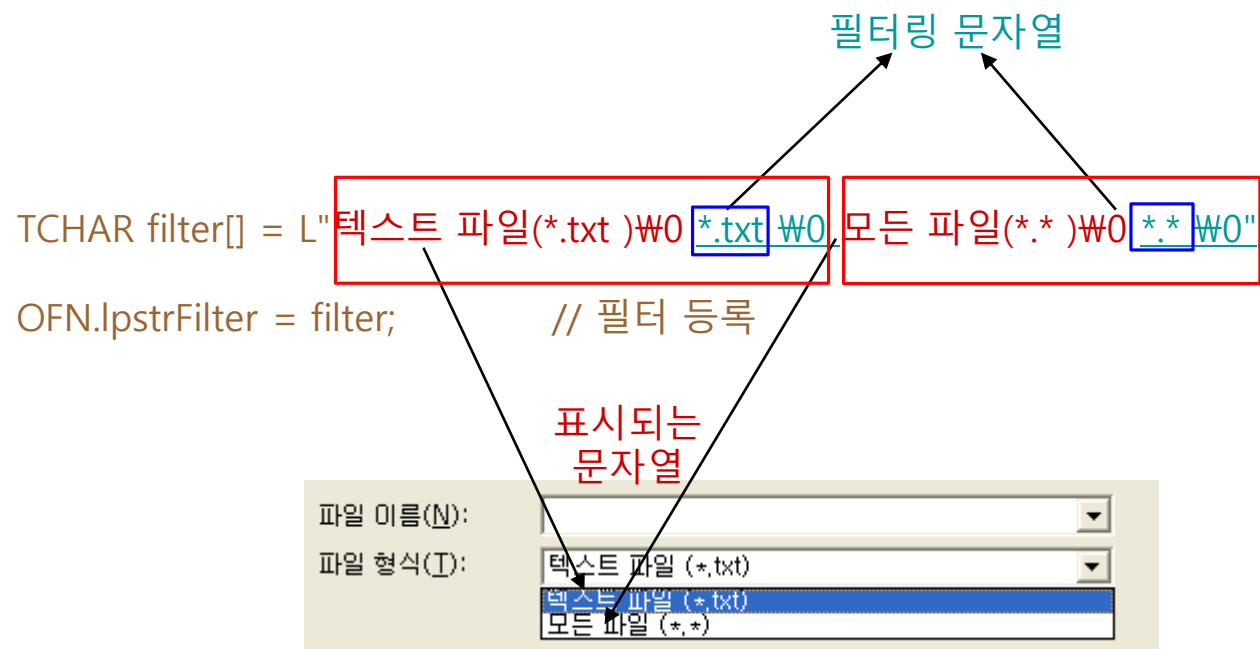
void **memset** (void *ptr, int value, size_t num);

- 어떤 메모리의 시작점부터 연속된 범위를 어떤 값으로 모두 지정할 때 사용

- ptr: 채우고자 하는 메모리의 시작 포인터 (주소값)
- value: 메모리에 채우고자 하는 값
- num: 채우고자 하는 바이트의 수 (메모리 크기)

필터 지정 방법

- 필터의 용도
 - 표시되는 파일이름을 걸러 줌
 - 정의시 **공문자** 삽입 안 하도록
 - 매 필터마다 널 문자로 종료하며 하나의 필터는 "파일형식W0필터W0"로 표시한다.
 - 한 개의 필터에 여러 개의 패턴 지정하려면 ; (**세미콜론**)으로 연결



파일 열기

- 사용 예) 파일 열기 함수 사용하기
 - 메뉴를 사용하여 파일 열기 공용 대화상자 열기

OPENFILENAME OFN;

//--- 파일 관련 구조체 선언

TCHAR str[100], lpstrFile[100] = L"";

TCHAR filter[100] = L"소스 파일(*.cpp)W0*.cppW0헤더 파일(*.h)W0*.hW0문서 파일(*.txt; *.doc) W0*.txt;*.docW0";

switch (iMsg)

{

case WM_COMMAND:

switch(LOWORD(wParam)) {

case ID_FILEOPEN:

memset(&OFN, 0, sizeof(OPENFILENAME));

OFN.lStructSize = sizeof(OPENFILENAME);

OFN.hwndOwner = hwnd;

OFN.lpstrFilter = **filter**;

OFN.lpstrFile = lpstrFile;

OFN.nMaxFile = 256;

OFN.lpstrInitialDir = L".";

//--- 메뉴 선택

//--- 구조체 초기화

//--- 초기 디렉토리

if (**GetOpenFileName** (&OFN)!=0) {

//--- 파일 열기 함수 호출

wsprintf (str, L"%s 파일을 여시겠습니까 ?", OFN.lpstrFile);

MessageBox (hwnd, str, L"열기 선택", MB_OK);

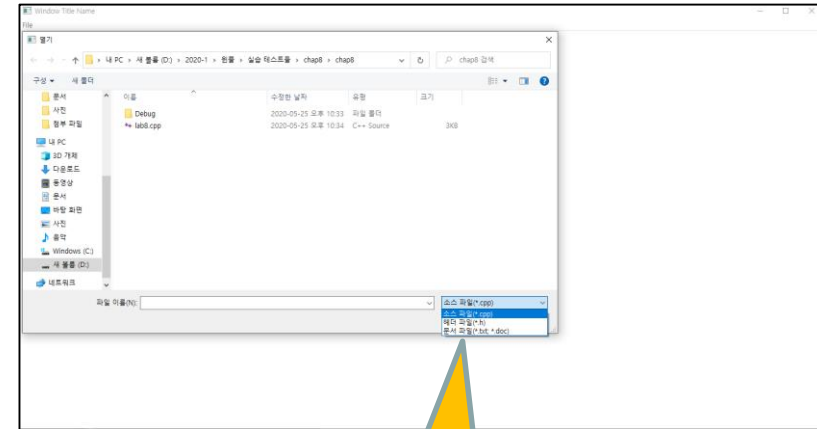
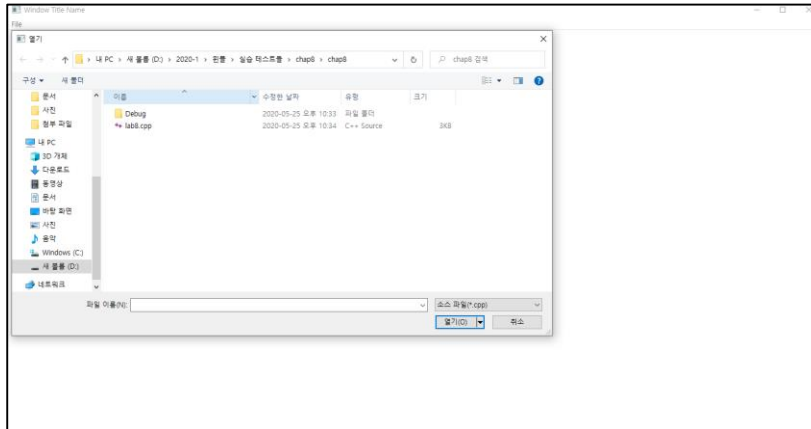
}

break;

*.txt와 *.doc 파일

파일 열기

- 결과 화면



필터 설정: *.cpp
*.h
*.hwp *.doc

파일 저장하기

- 사용 예) 파일 저장하기 함수 사용하기
 - 메뉴를 사용하여 파일 저장하기 공용 대화상자 열기

```
OPENFILENAME SFN;                                     //--- 파일열기와 저장하기는 동일한 구조체 사용
TCHAR str[100], lpstrFile[100] = L"";
TCHAR filter[100] = L"소스 파일(*.cpp)₩0*.cpp₩0헤더 파일(*.h)₩0*.h₩0문서 파일(*.txt; *.doc) ₩0*.txt;*.doc₩0";

switch (iMsg)
{
case WM_COMMAND:
    switch(LOWORD(wParam)) {
        case ID_FILESAVE:                                //--- 메뉴 선택
                                                            //--- 초기화
            memset (&OFN, 0, sizeof(OPENFILENAME));
            SFN.lStructSize = sizeof(OPENFILENAME);
            SFN.hwndOwner = hwnd;
            SFN.lpstrFilter = filter;
            SFN.lpstrFile = lpstrFile;
            SFN.nMaxFile = 256;
            SFN.lpstrInitialDir = ".";

            if (GetSaveFileName (&SFN)!=0) {              //--- 파일에 저장하기 함수 호출
                wsprintf (str, L"%s 파일에 저장하시겠습니까 ?", SFN.lpstrFile);
                MessageBox (hwnd, str, L " 저장하기 선택", MB_OK);
            }
            break;
```

2) 공용 대화상자: 폰트 선택하기

- 폰트 선택하기 처리절차
 - CHOOSEFONT 구조체 할당
 - LOGFONT 구조체 변수 연결
 - 폰트대화상자 띄우기 → 폰트정보 획득
 - 폰트 만들어 사용하기

CHOOSEFONT FONT;
LOGFONT LogFont;

```
FONT.lStructSize = sizeof(CHOOSEFONT);    //--- 구조체 크기
FONT.hwndOwner = hwnd;                    //--- 윈도우 핸들
FONT.lpLogFont = &LogFont;                //--- LOGFONT 구조체 변수 연결
FONT.Flags = CF_EFFECTS | CF_SCREENFONTS; //--- 폰트대화상자 옵션
ChooseFont (&FONT)                        //--- 폰트대화상자 띄우기

hFont = CreateFontIndirect(&LogFont);      //--- 선택된 폰트 핸들 생성
OldFont = (HFONT)SelectObject(hdc, hFont); //--- 폰트 사용
```


폰트 선택하기

- CHOOSEFONT 구조체

```
typedef struct {  
    DWORD      IStructSize;           //--- 구조체 크기  
    HWND       hwndOwner;            //--- 메인윈도우 핸들  
    HDC        hDC;                  //--- 메인 DC 핸들  
    LPLOGFONT   lpLogFont;           //--- LOGFONT 구조체 변수 값  
                                           //--- (글꼴 선택하면 설정된다.)  
    int        iPointSize;           //--- 선택한 글꼴의 크기 (글꼴 선택하면 설정된다)  
    DWORD      Flags;                //--- 글꼴 상자 초기화  
    COLORREF    rgbColors;            //--- 선택한 글꼴의 색상 정보 저장  
    LPARAM      lCustData;  
    LPCFHOOKPROC lpfnHook;  
    LPCTSTR     lpTemplateName;  
    HINSTANCE    hInstance;  
    LPSTR       lpszStyle;  
    WORD        nFontType;           //--- 선택한 글꼴을 가리키는 필드  
    int         nSizeMin;  
    int         nSizeMax;  
} CHOOSEFONT, *LPCHOOSEFONT;
```

COLORREF **SetTextColor** (HDC hdc, COLORREF crColor);
디바이스 컨텍스트에 이미 등록되어 있던 텍스트 색상값을 반환

- hdc: 변경할 디바이스 컨텍스트
- Color: 변경할 색상

폰트 선택하기

- LOGFONT 구조체

```
typedef struct {  
    LONG        lfHeight;           //--- 논리적 크기의 글꼴의 높이를 나타내는 정수  
    LONG        lfWidth;            //--- 글꼴의 너비  
    LONG        lfEscapement;       //--- 글꼴의 굽기 지정 (0 ~ 100 사이의 정수)  
    LONG        lfOrientation;      //--- 이탤릭 체 (TRUE/FALSE)  
    LONG        lfWeight;           //--- 글자에 밑줄 (TRUE/FALSE)  
    BYTE        lfItalic;           //--- 글자에 취소선 (TRUE/FALSE)  
    BYTE        lfUnderline;        //--- 글자에 취소선 (TRUE/FALSE)  
    BYTE        lfStrikeOut;        //--- 글자에 취소선 (TRUE/FALSE)  
    BYTE        lfCharSet;          //--- 문자 배열로 글꼴 이름 저장  
    BYTE        lfOutPrecision;     //--- 문자 배열로 글꼴 이름 저장  
    BYTE        lfQuality;          //--- 문자 배열로 글꼴 이름 저장  
    BYTE        lfPitchAndFamily;   //--- 문자 배열로 글꼴 이름 저장  
    TCHAR       lfFaceName[LF_FACESIZE];  
} LOGFONT;
```

폰트 선택하기

- 폰트 다루기 함수들

```
HFONT CreateFont ( int nHeight, int nWidth,int nEscapement, int nOrientation, int fnWeight,  
    DWORD fdwItalic, DWORD fdwUnderline, DWORD fdwStrikeOut, DWORD fdwCharSet,  
    DWORD fdwOutputPrecision,DWORD fdwClipPrecision, DWORD fdwQuality, DWORD  
    fdwPitchAndFamily, LPCTSTR lpzFace );
```

- 인수가 지정하는 특성에 가장 일치하는 논리 폰트를 생성하는 함수

```
HFONT CreateFontIndirect (CONST LOGFONT *lpf);
```

- LOGFONT 구조체가 지정하는 특성의 논리 폰트를 생성하는 함수

```
BOOL ChooseFont (LPCHOOSEFONT lpcf);
```

- 폰트 공용 대화상자를 여는 함수

폰트 선택하기

- 사용 예)

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

```
{
    CHOOSEFONT                FONT;
    HFONT                     hFont, OldFont;
    static LOGFONT            LogFont;
    static COLORREF           fColor;

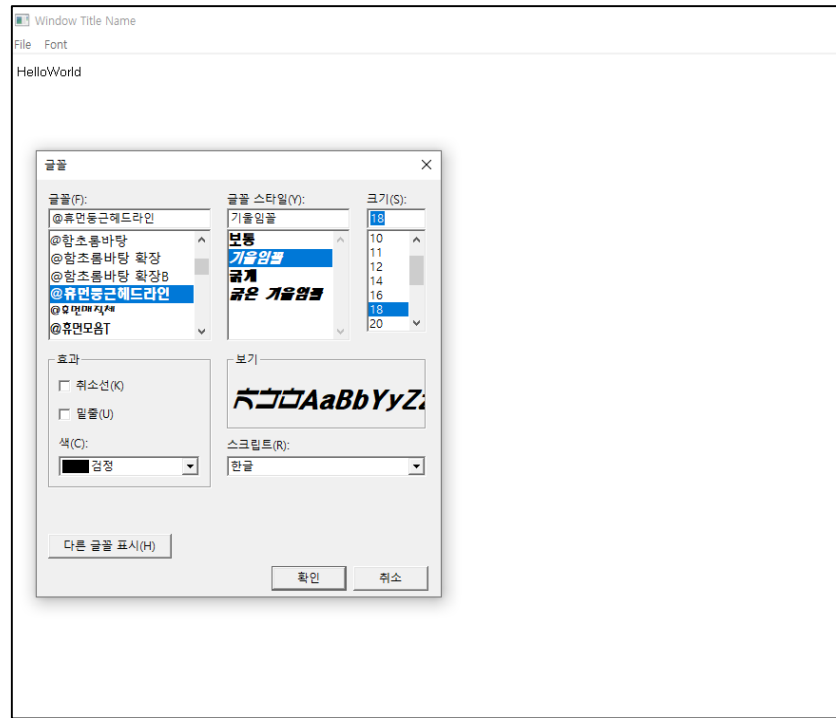
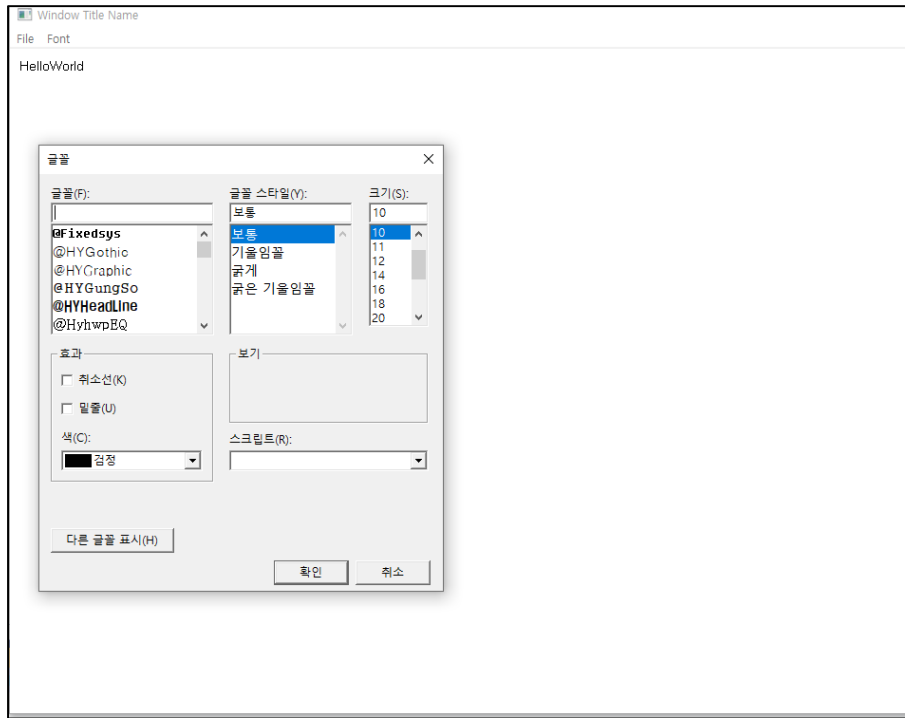
    switch (iMsg)
    {
        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                case ID_FONTDLG:
                    memset (&FONT, 0, sizeof(CHOOSEFONT));
                    FONT.lStructSize = sizeof(CHOOSEFONT);
                    FONT.hwndOwner = hwnd;
                    FONT.lpLogFont= &LogFont;
                    FONT.Flags = CF_EFFECTS | CF_SCREENFONTS;

                    if (ChooseFont(&FONT)!=0) {
                        fColor = FONT.rgbColors;
                        InvalidateRect (hwnd, NULL, TRUE);
                    }
                    break;

                case WM_PAINT:
                    hdc = BeginPaint(hwnd, &ps);
                    hFont = CreateFontIndirect(&LogFont);
                    OldFont = (HFONT) SelectObject (hdc, hFont);
                    SetTextColor(hdc, fColor);
                    TextOut(hdc, 10, 10, L"HelloWorld", 10);
                    SelectObject(hdc, OldFont);
                    DeleteObject(hFont);
                    EndPaint(hwnd, &ps);
                    break;
            }
        }
    }
}
```

폰트 선택하기

- 결과 화면



3) 공용 대화상자: 색상 선택하기

- 색상선택하기 처리절차
 - CHOOSECOLOR 구조체 할당
 - "사용자 지정 색" 만들기
 - 색상 대화상자 띄우기-> 색상 정보 획득

CHOOSECOLOR COLOR;

static COLORREF tmp[16], color;

For (int i=0; i<16; i++)

tmp[i] = 사용자 지정 색상;

memset(&COLOR, 0, sizeof(CHOOSECOLOR));

COLOR.lStructSize = sizeof(CHOOSECOLOR);

COLOR.hwndOwner = hwnd;

COLOR.lpCustColors = tmp;

COLOR.Flags = CC_FULLOPEN;

ChooseColor (&COLOR);

//--- COLOR.rgbResult에 색상정보 저장됨

BOOL ChooseColor (LPCHOOSECOLOR color);

- 색상 공용 대화상자를 여는 함수

색상 선택하기

- CHOOSECOLOR 구조체

```
typedef struct {  
    DWORD lStructSize;           //--- 구조체 크기  
    HWND hwndOwner;             //--- 메인 윈도우 핸들  
    HWND hInstance;  
    COLORREF rgbResult;        //--- 사용자가 대화상자에서 선택한 색상 정보  
    COLORREF *lpcustColors;      //--- 색 대화상자에 사용자 지정색에 채울 색 정보 목록 (16가지)  
    DWORD Flags;                //--- 색 대화상자 초기화 하는데 사용한 플래그  
    LPARAM lcustData;  
    LPCCHOOKPROC lpfnHook;  
    LPCTSTR lpTemplateName;  
} CHOOSECOLOR, *LPCHOOSECOLOR;
```

색상 선택하기

- 사용 예)

CHOOSECOLOR COLOR;

```
static COLORREF tmp[16], color;  
HBRUSH hBrush, OldBrush;  
int i;
```

case WM_COMMAND:

```
switch(LOWORD(wParam))
```

```
{
```

case ID_COLORDLG:

```
for(i=0;i<16;i++)
```

```
tmp[i] = RGB (rand()%256, rand()%256, rand()%256);
```

```
memset (&COLOR, 0, sizeof(CHOOSECOLOR));
```

```
COLOR.lStructSize = sizeof(CHOOSECOLOR);
```

```
COLOR.hwndOwner = hwnd;
```

```
COLOR.lpCustColors = tmp;
```

```
COLOR.Flags = CC_FULLOPEN;
```

```
if (ChooseColor (&COLOR)!=0) {
```

```
color = COLOR.rgbResult;
```

```
InvalidateRect (hwnd, NULL, TRUE);
```

```
}
```

```
break;
```

```
}
```

```
break;
```

case WM_PAINT:

```
hdc = BeginPaint(hwnd, &ps);
```

```
hBrush = CreateSolidBrush (color);
```

```
OldBrush = (HBRUSH)SelectObject(hdc, hBrush);
```

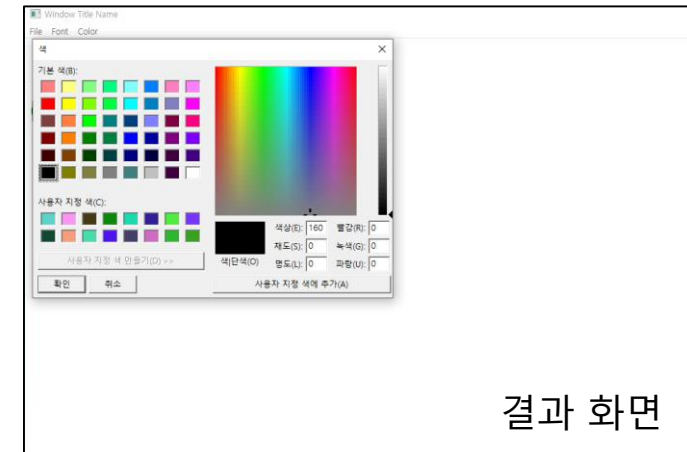
```
Ellipse(hdc, 10, 10, 200, 200);
```

```
SelectObject(hdc, OldBrush);
```

```
DeleteObject(hBrush);
```

```
EndPaint(hwnd, &ps);
```

```
break;
```



결과 화면

3. PeekMessage () 함수

- 지금까지 우리가 사용했던 메시지루프

```
while( GetMessage ( &msg, NULL, 0, 0 ) ){  
    TranslateMessage ( &msg );  
    DispatchMessage ( &msg );  
}
```

- GetMessage()함수는

- 메시지 큐에서 메시지를 읽은 후 이 메시지를 큐에서 제거한다.
- 메시지 큐에 대기중인 메시지가 없을 경우 메시지가 전달될 때까지 리턴하지 않고 무한히 대기한다.
- 특별한 일을 하지 않고 대기하는 시간에 다른 일을 하려면 PeekMessage()함수를 사용하는 것이 좋다.

PeekMessage () 함수

- PeekMessage() 함수는
 - GetMessage()함수처럼 메시지 큐에서 메시지를 읽는다.
 - 읽은 메시지를 무조건 제거하지 않으며 큐가 비어 있을 경우 대기하지 않고 곧바로 리턴한다.
 - 메시지를 읽지 않고 단순히 메시지가 있는지 확인만 할 수 있으며 이런 특성은 백그라운드(background) 작업에 적절하다.
- PeekMessage()함수로 메시지 루프를 구현했을 경우
 - WM_QUIT메시지에 대한 예외적인 처리를 반드시 해주어야 한다.
 - GetMessage() 함수: WM_QUIT메시지를 받으면 FALSE를 리턴하여 무한 메시지 루프를 빠져나올 수 있다.
 - PeekMessage() 함수: 메시지 존재 여부만 알려주므로 무한 메시지 루프를 빠져 나올 수 없다.

PeekMessage () 함수

- 두 함수의 프로토타입
 - BOOL GetMessage (LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax);
 - lpMsg: 메시지
 - hWnd: 메시지를 받을 윈도우 핸들
 - wMsgFilterMin: 조사할 메시지의 최소값
 - wMsgFilterMax: 조사할 메시지의 최대값
 - 리턴값: 조사한 메시지가 WM_QUIT 메시지이면 false 리턴, 그 외의 메시지이면 true 를 리턴
 - BOOL PeekMessage (LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax, UINT wRemoveMsg);
 - wRemoveMsg: 메시지 처리 방법 지정 플래그
 - PM_NOREMOVE: 메시지를 읽은 후 큐에서 메시지를 제거하지 않는다.
 - PM_REMOVE: 메시지를 읽은 후 큐에서 메시지를 제거한다.
 - 리턴값: 메시지 큐에 메시지가 있으면 0이 아닌 값을 리턴, 메시지가 없으면 0을 리턴

PeekMessage () 함수

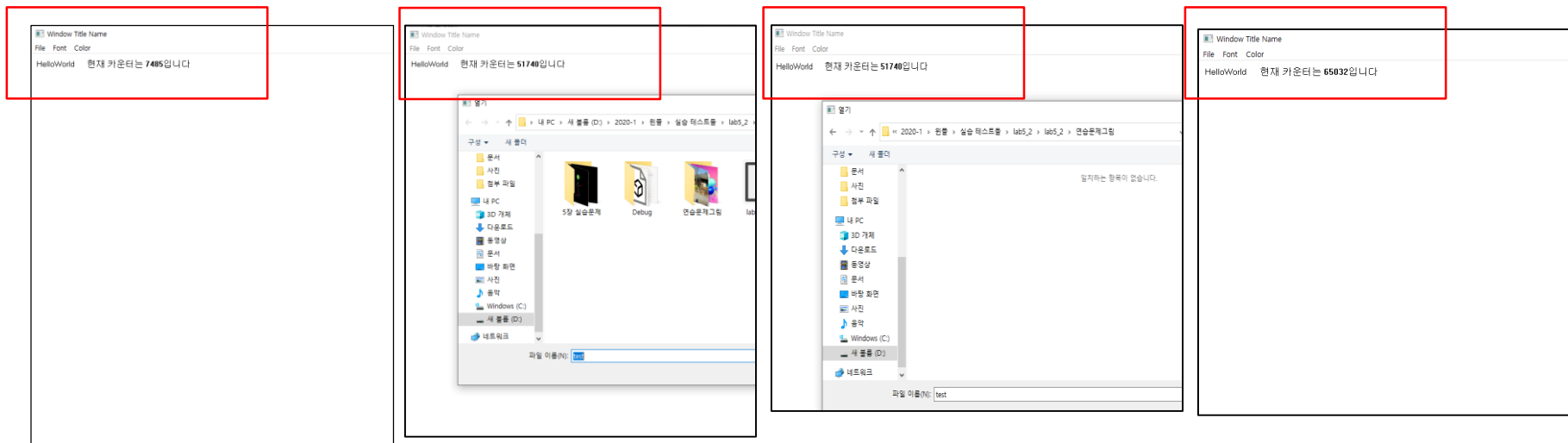
- PeekMessage()함수를 사용한 메시지 루프는 다음과 같이 구현한다.

```
while (1) {  
    if ( PeekMessage (&msg, NULL, 0, 0, PM_REMOVE ) ){  
        if ( msg.message == WM_QUIT )  
            break;  
  
        TranslateMessage (&msg);  
        DispatchMessage (&msg);  
    }  
}
```

PeekMessage () 함수

- 사용 예) 카운트를 세는 작업을 백 그라운드로 하고 있다.

```
while (1)
{
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
        if (msg.message == WM_QUIT)
            break;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else {
        count++;
        wsprintf (str, _T("현재 카운터는 %d입니다"), count);
        TextOut (hdc, 10, 10, str, lstrlen(str));
    }
}
```



4. GetAsyncKeyState () 함수 / GetKeyState () 함수

- 키보드의 키가 눌렸는지 체크하는 함수
 - SHORT **GetAsyncKeyState** (int vKey);
 - 호출된 시점에서 키 상태를 조사 → 메시지 큐를 거치지 않고 바로 리턴 → 키 입력을 바로 처리
 - SHORT **GetKeyState** (int vKey):
 - 호출된 시점에서 메시지 큐를 거친다 → 메시지 발생 후의 상태를 리턴 → 키보드 메시지 처리 루틴 내에서 사용
 - 키 입력에 바로 반응 해줘야 하는 프로그램에서는 **GetAsyncKeyState** 함수를 사용하는 편이 좋다.

값	가상키 코드	설명	값	가상키 코드	설명	값	가상키 코드	설명
0x01	VK_LBUTTON		0x11	VK_CONTROL	Ctrl	0x24	VK_HOME	Home
0x02	VK_RBUTTON		0x12	VK_MENU	Alt	0x25	VK_LEFT	←
0x03	VK_CANCEL		0x13	VK_PAUSE	Pause	0x26	VK_UP	↑
0x04	VK_MBUTTON		0x14	VK_CAPITAL	Caps lock	0x27	VK_RIGHT	→
0x08	VK_BACK	Backspace	0x1B	VK_ESCAPE	esc	0x28	VK_DOWN	↓
0x09	VK_TAB	Tab	0x20	VK_SPACE	Space Bar			
0x0C	VK_CLEAR		0x21	VK_PRIOR	Page Up			
0x0D	VK_RETURN	Enter	0x22	VK_NEXT	Page Down			
0x10	VK_SHIFT	shift	0x23	VK_END	End			

GetAsyncKeyState () 함수 / GetKeyState () 함수

- SHORT GetAsyncKeyState (int vKey);
 - vKey: 가상키 코드 값, 확인하고자 하는 키를 입력
 - 리턴 값:

반환값	내용
0 (0x0000)	이전에 누른 적이 없고 호출 시점에 안 눌린 상태
0x8000 (최상위 비트 세팅)	이전에 누른 적이 없고 호출 시점에서 눌린 상태
0x8001 (최상위, 최하위 비트)	이전에 누른 적이 있고 호출 시점에 눌린 상태
1 (0x0001) (최하위 비트)	이전에 누른 적이 있고 호출 시점에 안 눌린 상태

- 함수 사용

```
if (GetAsyncKeyState (VK_RETURN)) {  
    //--- 필요한 작업 처리  
}
```

전에 눌러졌다면, 0이 아닌 값이 리턴된다.

또는

```
if (GetAsyncKeyState (VK_RETURN) & 0x8000 ) {  
    //--- 필요한 작업 처리  
}
```

정확한 시점에 키가 눌러진 상태를 체크할 수 있다

GetAsyncKeyState () 함수 / GetKeyState () 함수

- 함수 사용 예)

- 스페이스 키 현재 상태 체크

```
case WM_KEYDOWN:
```

```
    if ( GetAsyncKeyState (VK_SPACE) & 0x8000)    //-- 현재 시점의 키눌림 상태를 체크
```

```
        //-- 스페이스키가 눌러짐
```

```
    Else
```

```
        //-- 스페이스키가 안 눌러짐
```

- 동시 키 입력 상태 체크하기: left 키와 컨트롤 키를 함께 누른 경우

```
case WM_KEYDOWN:
```

```
    if ( (wParam == VK_LEFT) && (GetAsyncKeyState (VK_CONTROL) & 0x8000) )    //-- Left 키와 control 키 눌림 상태 체크
```

```
        //-- left 키와 컨트롤 키가 눌러졌음
```

```
    else
```

```
        //-- 두 키가 같이 눌러지지 않았음
```


GetAsyncKeyState () 함수 / GetKeyState () 함수

- SHORT GetKeyState (int vKey):
 - vKey: 가상키 코드 값, 확인하고자 하는 키를 입력
 - 리턴 값
 - 음수값: 해당 키가 눌린 상태
 - 음수가 아닐 경우: 해당 키가 눌리지 않은 상태
 - 함수 사용 예)

```
case WM_KEYDOWN:  
    if ( (wParam == VK_LEFT) && (GetKeyState(VK_CONTROL) < 0) ) {    //-- 컨트롤키와 왼쪽화살표키 상태 체크  
        //-- 컨트롤키와 왼쪽 화살표키가 눌렸음  
    }  
    else {  
        //-- 두키가 동시에 눌러지지 않았음  
    }
```

GetAsyncKeyState () 함수 / GetKeyState () 함수

- GetAsyncKeyState 와 GetKeyState 차이점
 - GetAsyncKeyState 함수는 "키가 눌렸는가"와 "언제부터 눌렸는가" 를 알아낼 때 사용
 - 키가 눌렸을 때 GetAsyncKeyState는 0x8000 bit가 1이된다.
 - 그리고, 이전에 GetAsyncKeyState가 호출되었을 때부터 이번에 GetAsyncKeyState가 호출될 때까지
 - 중간에 끊기지 않고 계속 눌러있는 상태라면 0x0001 bit (최하위 비트)는 0이 되고, 그렇지 않은 경우는 1이 된다.
 - 예를 들어 컨트롤 키의 상태가
 - 1) CTRL 키가 눌린 상태이다.
 - 2) GetAsyncKeyState(VK_CONTROL)를 호출 → 0x8001을 리턴
 - 3) GetAsyncKeyState(VK_CONTROL)을 한번 더 호출하면 → 0x8000을 리턴
 - 4) CTRL 키가 눌린 상태이다.
 - 5) GetAsyncKeyState(VK_CONTROL)를 호출 → 0x8001을 리턴
 - 6) CTRL 키를 뗐다가 다시 눌렀다.
 - 7) GetAsyncKeyState(VK_CONTROL)을 한번 더 호출하면 → 0x8001을 리턴
 - 키가 눌리지 않았을 때, GetAsyncKeyState는 항상 0x0000을 리턴한다.
- 호출된 시점에서 키 상태를 조사, 메시지 큐를 거치지 않고 바로 리턴해주므로 키 입력을 바로 처리해줄 수 있다.
- GetKeyState 함수는 "키가 눌렸는가" 와 "키의 토글 상태가 무엇인가" 를 알아낼 때 사용
- 현재 키보드의 상태값을 알아내길 원한다면 GetAsyncKeyState 사용하는 것을 추천

5. 사운드 이용하기: PlaySound 함수

- 윈도우에서 지원하는 사운드 재생 함수
 - BOOL PlaySound (LPCSTR pszSound, HMODULE hmode, DWORD fdwSound);
 - 파일 이름 또는 리소스로 사운드 재생
 - wav 파일 재생
 - 링크 추가: [winmm.lib](#) 링크 추가 (프로젝트 속성 → 링커 → 명령줄)
 - 헤더파일 추가: [#include <mmsystem.h>](#)
 - 사용예)
 - PlaySound ("Sample.wav", NULL, SND_ASYNC|SND_LOOP);
 - PlaySound (NULL, NULL, NULL); //--- 사운드 재생을 정지할 때

BOOL **PlaySound** (LPCSTR pszSound, HMODULE hmode, DWORD fdwSound);

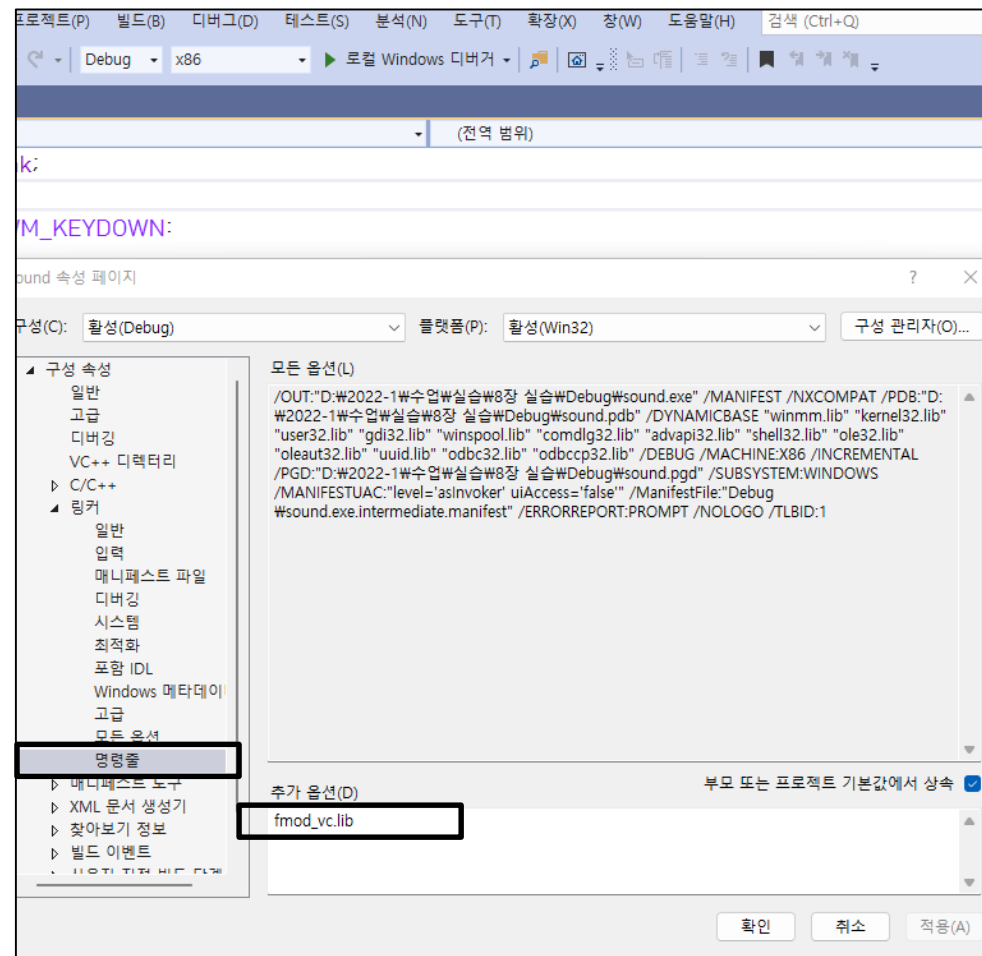
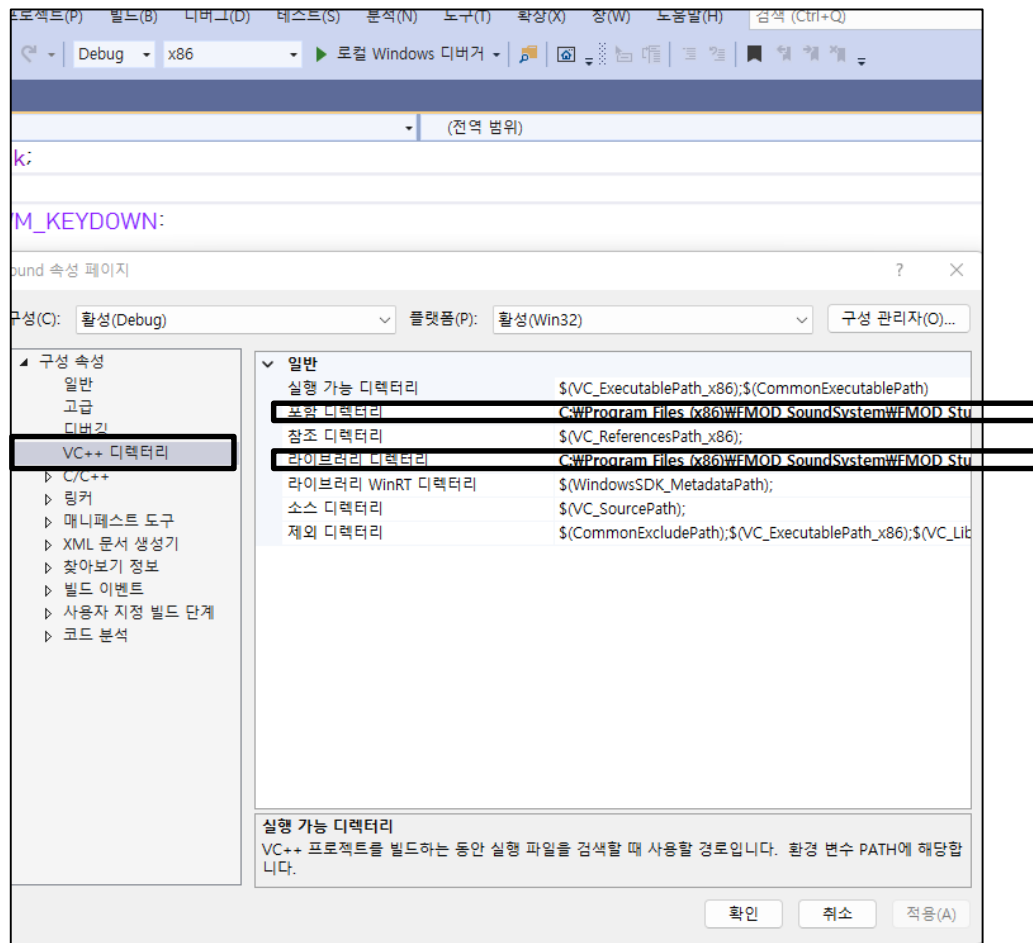
- 윈도우가 지원하는 사운드 재생 함수

- lpszSound: 재생할 파일 명, NULL이면 소리 재생 중지
- hmode: 읽을 수 있는 리소스를 포함한 실행가능한 파일 핸들, 세 번째 인자가 SND_RESOURCE가 명시되지 않으면 NULL로 지정
- fdwSound: 사운드의 연주 방식과 연주할 사운드의 종류 정의
- SND_ASYNC: 비동기화된 연주, 연주를 시작한 직후 다른 작업을 바로 시작할 수 있다. 연주를 중지하려면 lpszSound를 null값으로 하여 함수를 호출
- SND_LOOP: 지정한 함수를 반복적으로 계속 연주 (SND_ASYNC와 함께 사용)
- SND_SYNC: 동기화된 연주로 소리 이벤트 완료 후 함수 반환

6. 사운드 이용하기: FMOD 사운드 시스템

- FMOD 사운드 엔진:
 - FireLight Technologies(fmod.com)에서 만든 음향 [미들웨어](#)
 - fmod.com에서 "download ➔ FMOD Engine ➔ Windows (2.00.03 버전)" 다운로드
 - 설치
 - 프로젝트에 설정하기
 - 프로젝트 속성에서
 - » VC++ 디렉토리 → 포함 디렉터리 → C:\Program Files (x86)\FMOD SoundSystem\FMOD Studio API Windows\api\core\inc
 - » VC++ 디렉토리 → 라이브러리 디렉터리 → C:\Program Files (x86)\FMOD SoundSystem\FMOD Studio API Windows\api\core\lib\x86
 - » 프로젝트 속성 → 링커 → 입력 → 추가종속성 → fmod_vc.lib 추가
또는, 프로젝트 속성 → 링커 → 명령줄 → fmod_vc.lib 추가
 - 헤더파일 추가하기
 - » #include <fmod.h>
 - 프로젝트가 있는 폴더에 fmod.dll 추가하기
 - » C:\Program Files (x86)\FMOD SoundSystem\FMOD Studio API Windows\api\core\lib\x86\fmod.dll 파일 복사하기
 - 출력 가능한 사운드 형식
 - AIFF, ASF, ASX, DLS, FLAC, FSB, IT, M3U, MID, MOD, MP2, MP3, OGG, PLS, RAW, S3M, VAG, WAV, WMA, XM, XMA 등 대부분의 사운드 형식 사용 가능

FMOD 사운드 시스템



FMOD 사운드 시스템

- 사운드 시스템 생성과 초기화

- 시스템 변수 선언
- 시스템 객체 생성
- Fmod 시스템 초기화하기

- 사용 예)

```
FMOD_SYSTEM          *soundSystem;                //--- FMOD system 변수 선언
```

```
FMOD_System_Create (&soundSystem);                //--- FMOD system 객체 생성
```

```
FMOD_System_Init (soundSystem, 32, FMOD_INIT_NORMAL, NULL); //--- FMOD system 초기화
```

```
FMOD_RESULT FMOD_System_Create (FMOD_SYSTEM **system);
```

- FMOD 사운드 시스템을 생성한다
 - system: FMOD 시스템

```
FMOD_RESULT FMOD_System_Init (FMOD_SYSTEM *system, int MaxChannels, FMOD_INITFLAGS flags, void *ExtraDriverdata);
```

- FMOD 시스템 초기화
 - System: FMOD 시스템
 - MaxChannels: FMOD에서 사용될 최대 채널 수
 - flags: 시작할 때 FMOD 상태 (FMOD_INIT_NORMAL)
 - ExtraDriverdata: 출력 플러그인에 보내질 드라이버 (NULL로 설정하면 무시한다).

FMOD 사운드 시스템

- 사운드 객체 생성과 사운드 로딩
 - 사운드 객체 생성: 사운드와 채널 객체 선언
 - 사운드 객체는 사운드 파일과 1:1 대응
 - 출력할 사운드 개수만큼 선언하여 사용
- 사용 예)

```
FMOD_SOUND      *soundFile;           //--- 사운드 객체 선언
FMOD_CHANNEL     *channel;             //--- 채널 선언
```

```
FMOD_System_CreateSound (soundSystem, "bgm.mp3", FMOD_LOOP_NORMAL, 0, &soundFile);
```

```
FMOD_RESULT FMOD_System_CreateSound (FMOD_SYSTEM *system, const char *name_or_data,
    FMOD_MODE mode, FMOD_CREATESOUNDEXINFO *exinfo, FMOD_SOUND **sound);
```

- 사운드 객체 생성 함수
 - name_or_data: 파일 경로 및 이름
 - Mode: 사운드 모드
 - FMOD_LOOP_NORMAL: 사운드를 반복 출력
 - FMOD_DEFAULT: FMOD_LOOP_OFF | FMOD_2D | FMOD_HARDWARE, 1번 출력
 - exinfo: 사운드에 대한 추가적인 확장 정보를 위한 FMOD_CREATESOUNDEXINFO 포인터 (0으로 설정하면 무시한다)
 - sound: 새로 만든 sound 객체

FMOD 사운드 시스템

- 사운드 재생
 - 사운드 객체 (FMOD_SOUND *sound) 를 통해 읽혀진 사운드는 채널(channel)을 통해 재생
 - 사용 예)

```
FMOD_System_PlaySound (soundSystem, soundFile, NULL, 0, &Channel);    //--- 사운드 재생
FMOD_Channel_SetVolume (channel, 0.5);                                //--- 볼륨 조절하기
```

```
FMOD_RESULT FMOD_System_PlaySound ( FMOD_SYSTEM *system, FMOD_SOUND *SoundFile,
                                     FMOD_CHANNELGROUP *ChannelGroup, FMOD_BOOL paused, FMOD_CHANNEL **Channel);
```

- 사운드를 재생한다

- System: FMOD 시스템
- SoundFile: 재생할 사운드 파일
- ChannelGroup: 채널 그룹
- paused: 채널이 멈추었을 때 시작할지 아닐지 (true/false)
- Channel: 새롭게 재생될 채널 (NULL이면 무시한다)

```
FMOD_RESULT FMOD_Channel_SetVolume (FMOD_CHANNEL *Channel, float volume);
```

- 볼륨 조절하기

- volume: 0.0 ~ 1.0 사이의 볼륨

FMOD 사운드 시스템

- 사운드 정지
 - 채널을 통해 출력중인 사운드를 정지시킬 수 있다.

```
FMOD_RESULT FMOD_Channel_Stop (FMOD_CHANNEL *channel)  
– 시스템 닫기

- channel: 정지할 채널

```

- FMOD 해제하기
 - 사용 후 시스템을 해제한다.

```
FMOD_RESULT FMOD_System_Release (FMOD_SOUND **sound)  
– FMOD sound 객체 해제

- sound: 해제할 사운드 객체

```

```
FMOD_RESULT FMOD_System_Close (FMOD_SYSTEM **system)  
– 시스템 닫기

- system: 닫을 시스템

```

```
FMOD_RESULT FMOD_System_Release (FMOD_SYSTEM ** system)  
– 시스템 객체 해제하기

- system: 해제할 시스템

```

동시 키 입력을 통한 이펙트 음악 재생하기 샘플 코드

```
#include <fmod.h>

#define SOUND_COUNT 2
#define EFFECT_COUNT 2
#define CHANNEL_COUNT 2

FMOD_SYSTEM *System;
FMOD_SOUND *bgmSound[SOUND_COUNT];
FMOD_SOUND *effectSound[EFFECT_COUNT];
FMOD_CHANNEL *Channel[CHANNEL_COUNT];

void Sound_Setup()
{
    char string[100];

    FMOD_System_Create (&System);
    FMOD_System_Init (System, 10, FMOD_INIT_NORMAL, NULL);

    for (int i = 0; i < SOUND_COUNT; i++)
    {
        wsprintf (string, "sound%d.mp3", i);
        FMOD_System_CreateSound (System, string, FMOD_LOOP_NORMAL, 0, &bgmSound[i]);
    }

    FMOD_System_CreateSound (System, "effect0.mp3", FMOD_DEFAULT, 0, &effectSound[0]);
    FMOD_System_CreateSound (System, "effect3.wav", FMOD_DEFAULT, 0, &effectSound[1]);

    Channel[0] = Channel[1] = 0;
}
```

동시 키 입력을 통한 이펙트 음악 재생하기 샘플 코드

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
```

```
{
```

```
    HDC hdc;
```

```
    PAINTSTRUCT ps;
```

```
    static int x[2], y[2], sound, xDir;
```

```
    int i;
```

```
    RECT rect;
```

```
    static int nWidth, nHeight;
```

```
    switch (iMsg)
```

```
    {
```

```
        case WM_CREATE:
```

```
            GetClientRect(hWnd, &rect);
```

```
            nWidth = rect.right;    nHeight = rect.bottom;
```

```
            x[0] = 0;
```

```
            y[0] = 100;
```

```
            x[1] = x[0] + 20;
```

```
            y[1] = y[0] + 20;
```

```
            xDir = 3;
```

```
            Sound_Setup();
```

```
            SetTimer(hWnd, 1, 100, NULL);
```

```
            //--- object moving
```

```
            soundNum = 0;
```

```
        break;
```

```
        case WM_PAINT:
```

```
            hdc = BeginPaint(hWnd, &ps);
```

```
            Rectangle(hdc, x[0], y[0], x[1], y[1]);
```

```
            EndPaint(hWnd, &ps);
```

```
        break;
```

```
        case WM_TIMER:
```

```
            for (i = 0; i < 2; i++)
```

```
                x[i] += xDir;
```

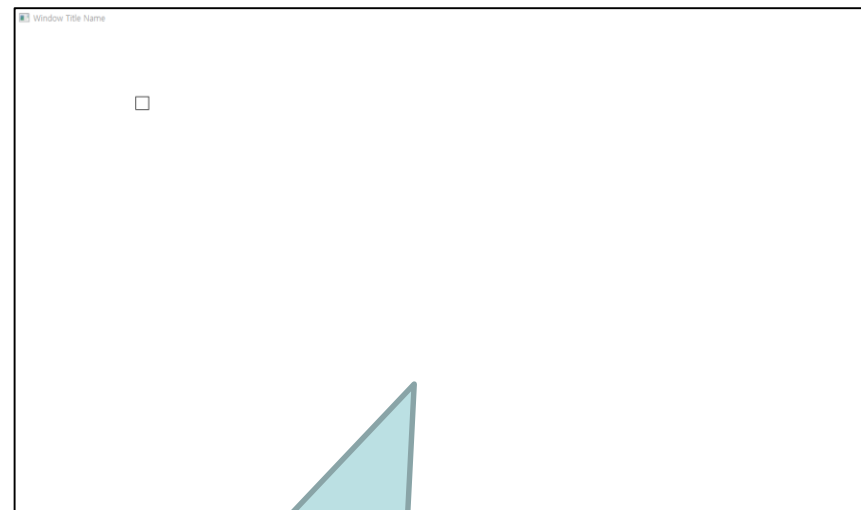
```
            //--- 원 이동
```

```
            if (x[0] > nWidth)    xDir *= -1;
```

```
            else if (x[0] < 0)    xDir *= -1;
```

```
            InvalidateRect(hWnd, NULL, true);
```

```
        break;
```



왼쪽 마우스버튼,
컨트롤-좌측 버튼,
컨트롤-오른쪽 버튼
키보드 p를 누르면 음악 재생

키보드 q 누르면 음악 중지

화면의 사각형은 좌우로 이동

동시 키 입력을 통한 이펙트 음악 재생하기 샘플 코드

```
case WM_KEYDOWN:
    if ((wParam == VK_LEFT) && (GetAsyncKeyState(VK_CONTROL)))
        FMOD_System_PlaySound (System, effectSound[0], 0, false, &Channel[0]);
    else if ((wParam == VK_RIGHT) && (GetKeyState(VK_CONTROL)<0))
        FMOD_System_PlaySound (System, effectSound[1], 0, false, &Channel[1]);
    break;

case WM_LBUTTONDOWN:
    FMOD_System_PlaySound (System, effectSound[0], 0, false, &Channel[1]);
    break;

case WM_CHAR:
    switch (wParam) {
        case 'p':
            soundNum++;
            soundNum %= SOUND_COUNT;
            FMOD_Channel_Stop (Channel[0]); //--- 채널0번에서 현재 생성되고 있는 사운드 종료
            FMOD_System_PlaySound (System, bgmSound[soundNum], 0, false, &Channel[0]); //--- 배경 사운드를 0번 채널에서 재생
            break;
        case 'q':
            FMOD_Channel_Stop (Channel[0]);
            FMOD_Channel_Stop (Channel[1]);
            break;
    }
    InvalidateRect(hWnd, NULL, FALSE);
    break;

case WM_DESTROY:
    for (i = 0; i < EFFECT_COUNT; i++) FMOD_Sound_Release(effectSound[i]);
    for (i = 0; i < SOUND_COUNT; i++) FMOD_Sound_Release(bgmSound[i]);
    FMOD_System_Release(System);
    KillTimer(hWnd, 1);
    PostQuitMessage(0);
    return 0;
}
return (DefWindowProc(hWnd, iMsg, wParam, lParam));
```

//--- control와 Left 키
//--- 이펙트 0번을 1번 채널에서 생성
//--- control와 Right 키
//--- 이펙트 1번을 1번 채널에서 생성

- 파일 입출력
- 공용 대화상자
- 유용한 함수들
 - 사운드
 - 동시 키입력
- 다음 시간
 - 6장, 7장 실습 채점
 - 8장 실습: 7-3번 실습에 데이터 저장, 읽기 기능 추가하기