8장 파일 입출력

2020년도 1학기 윈도우 프로그래밍

학습 목표

- 학습목표
  - 표준 입출력이 아닌 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 명시
  - 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

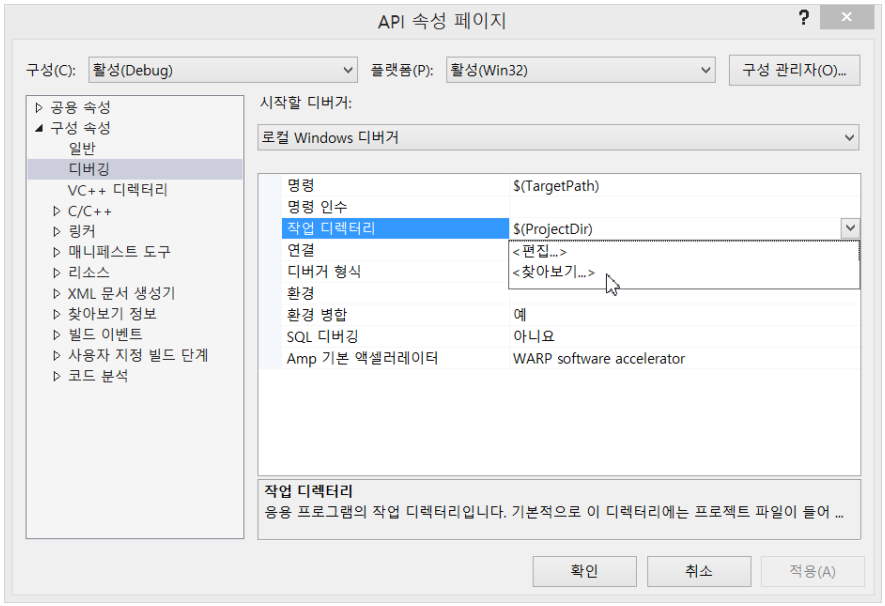
- 파일 생성 함수

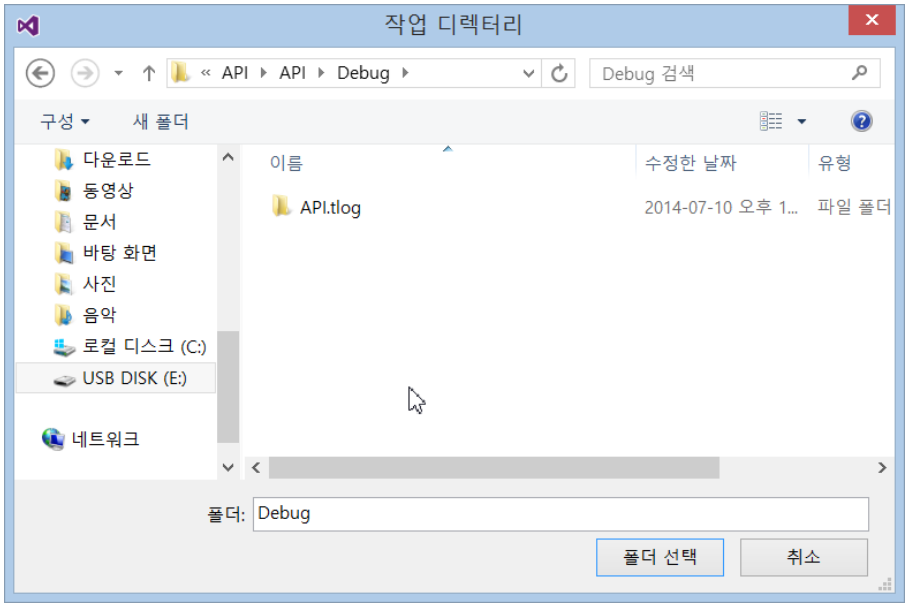
HANDLE **CreateFile** (LPCTSTR lpFileName, DWORD dwDesiredAccess,
DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes,
DWORD dwCreationDisposition, DWORD dwFlagsAndAttributes,
HANDLE hTemplateFile);

- 파일을 생성, 또는 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
- dwCreationDistribution:파일 생성 모드
  - 새로 만들기, 이미 있으면 에러 메시지: CREATE_NEW
  - 항상 새로 만들기, 파일이 있어도 파괴하고 새로 만듦: 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
- nFlagTemplate: 생성될 파일의 속성을 제공할 템플릿 파일

파일 생성/열기 함수

- 작업 디렉터리 위치 확인





파일 생성/열기 함수

- 파일 읽기 함수

```
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 생성/열기 함수

- 파일 쓰기 함수

```
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 함수

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

- 파일 닫기 함수
- hFile: 닫을 파일 핸들

임의 접근

- 파일 엑세스할 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 lpDistanceToMoveHight, DWORD dwMoveMethod);
```

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

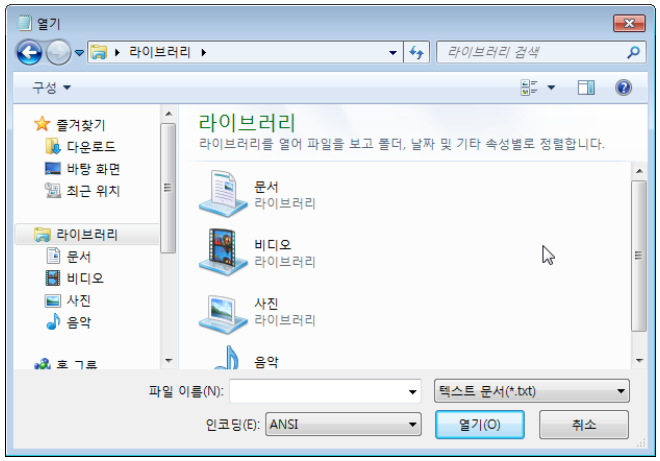
파일 입출력 예

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

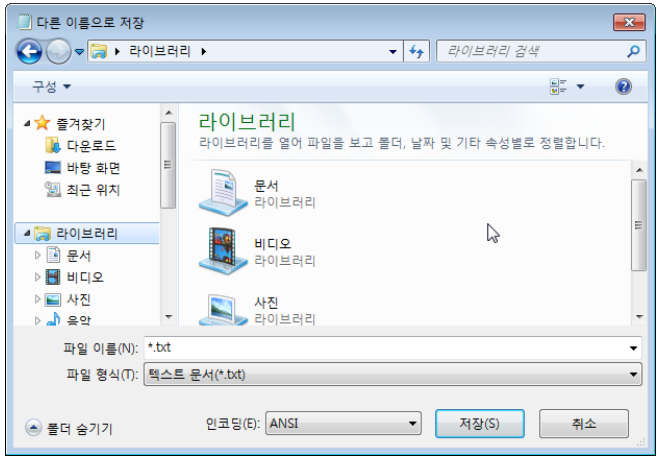
```
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    HANDLE hFile;
    TCHAR InBuff[1000];
    TCHAR OutBuff[1000] = L"₩nAPI 파일 입출력 테스트입니다.";
    int size = 1000, read_size;

    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

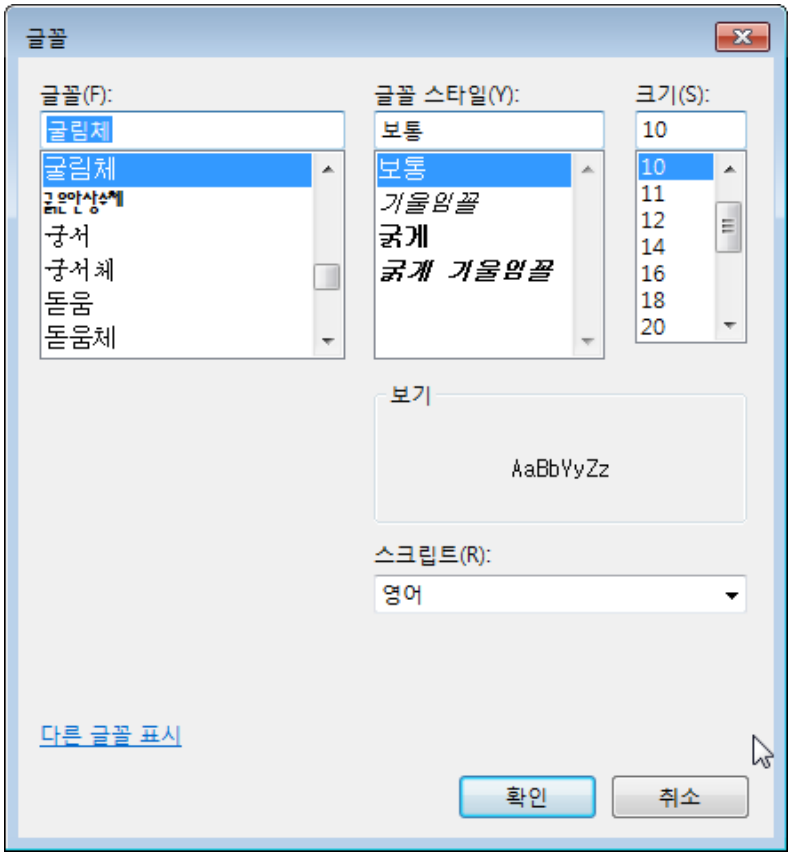
- 윈도우의 공용 대화상자
  - 구조체 설정 ➔ 함수 호출



파일 열기



파일 저장하기



글꼴 선택하기

1) 공용대화상자: 파일 열기

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

OPENFILENAME 구조체

```
typedef struct tagOFN{                                  //--- ofn
    DWORD   lStructSize;                                //--- 구조체 크기
    HWND                hwndOwner;                      //--- 오너 윈도우 핸들
    HINSTANCE           hInstance;                      //--- 인스턴스 핸들
    LPCTSTR             lpstrFilter;                    //--- 파일 형식 콤보 박스에 나타낼 필터
    LPTSTR              lpstrCustomFilter;              //--- 커스텀 필터를 저장하기 위한 버퍼
    DWORD               nMaxCustFilter;                 //--- 커스텀 필터 버퍼의 길이
    DWORD               nFilterIndex;                   //--- 파일 형식 콤보 박스에서 사용할 필터의 인덱스
    LPTSTR              lpstrFile;                      //--- 파일 이름 에디트에 처음 나타낼 파일명
                                                        //--- 최종적으로 선택된 파일이름이 저장된다.
    DWORD               nMaxFile;                       //--- lpstrFile 멤버의 길이
    LPTSTR              lpstrFileTitle;                 //--- 선택한 파일명을 리턴받기 위한 버퍼 (경로X)
    DWORD               nMaxFileTitle;                  //--- lpstrFileTitle 멤버의 길이
    LPCTSTR             lpstrInitialDir;                //--- 파일 찾기를 시작할 디렉토리
    LPCTSTR             lpstrTitle;                     //--- 대화상자의 캡션
    DWORD               Flags;                          //--- 대화상자의 모양과 동작을 지정하는 플래그
    WORD                nFileOffset;                    //--- lpstrFile 버퍼 내의 파일명 오프셋
    WORD                nFileExtension;                 //--- lpstrFile 버퍼 내의 파일 확장자 오프셋
    LPCTSTR             lpstrDefExt;                    //--- 디폴트 확장자
    DWORD               lCustData;                      //--- 훅 프로시저로 보낼 사용자 정의 데이터
    LPOFNHOOKPROC       lpfnHook;                       //--- 훅 프로시저명
    LPCTSTR             lpTemplateName;                 //--- 템플리트명
}OPENFILENAME;
```

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: 입력을 위한 OPENFILENAME 구조체 포인터

BOOL **GetSaveFileName** (LPOPENFILENAME lpsfn);
- 파일 입출력을 위해 파일 공용 대화상자를 열어 대상 파일을 입력받기 위해 호출되는 함수
- lpofn: 출력력을 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);
}
```

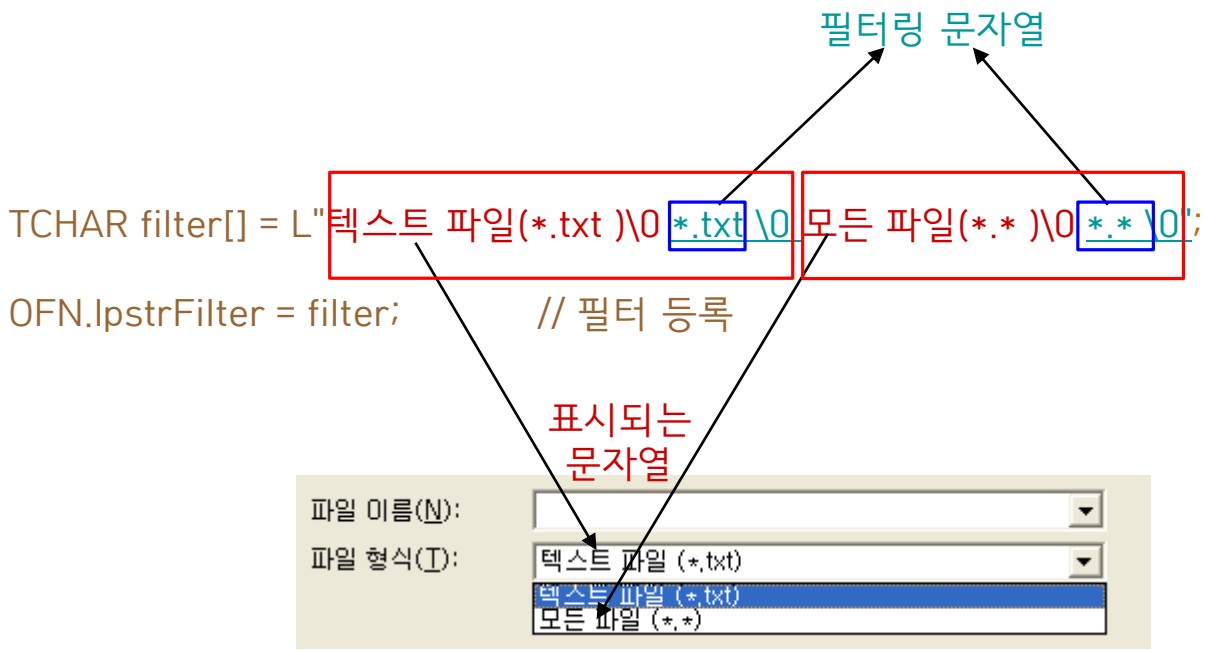
void **memset** (void *ptr, int value, size_t num);

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 **"파일형식\0필터\0"**로 표시한다.
  - 한 개의 필터에 여러 개의 패턴 지정하려면 **; (세미콜른)**로 연결



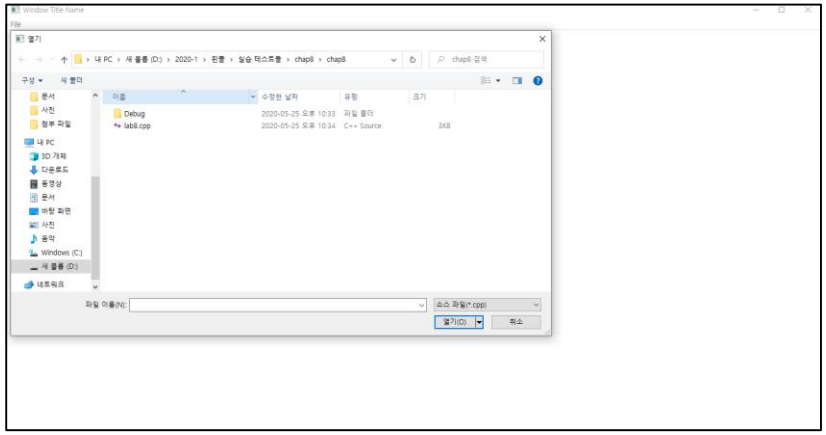
파일 열기

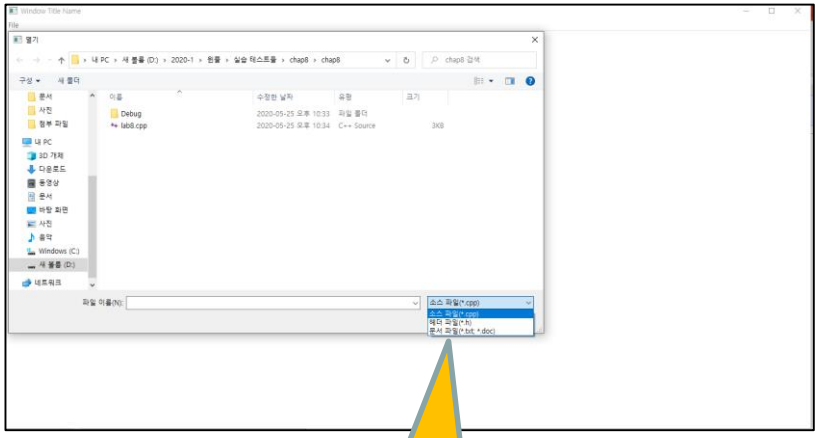
- 사용 예) 파일 열기 함수 사용하기
  - 메뉴를 사용하여 파일 열기 공용 대화상자 열기

```
OPENFILENAME OFN;                                   //--- 파일 관련 구조체 선언
TCHAR str[100], lpstrFile[100] = L"";
TCHAR filter[100] = L"소스 파일(*.cpp)₩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_FILEOPEN:                          //--- 메뉴 선택
               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 = L".";          //--- 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;
```

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
               memset (&OFN, 0, sizeof(OPENFILENAME));              //--- 초기화
               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          lStructSize;          //---  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;
        WORLD          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;
        LONG        lfOrientation;
        LONG        lfWeight;                   //--- 글꼴의 굵기 지정 (0 ~ 100 사이의 정수)
        BYTE        lfIatlic;                   //--- 이탤릭 체 (TRUE/FALSE)
        BYTE        lfUnderline;                //--- 글자에 밑줄 (TRUE/FALSE)
        BYTE        lfStrikeOut;                //--- 글자에 취소선 (TRUE/FALSE)
        BYTe        lfCharSet;
        BYTE        lfOutPrecision;
        BYTE        lfQuality;
        BYTE        lfPitchAndFamily;
        TCHAR       lfFaceName[LF_FACESIZE]; //---  문자 배열로 글꼴 이름 저장
} LOGFONT;
```

폰트 선택하기

- 폰트 다루기 함수들

HFONT **CreateFont** (int nHeight, int nWidth,int nEscapement, int nOrientation, int fnWeight, DWORD fdwItalic, DWORDfdwUnderline, DWORD fdwStrikeOut, DWORD fdwCharSet, DWORD fdwOutputPrecision,DWORD fdwClipPrecision, DWORD fdwQuality, DWORD fdwPitchAndFamily, LPCTSTRlpszFace);

- 인수가 지정하는 특성에 가장 일치하는 논리 폰트를 생성하는 함수

HFONT **CreateFontIndirect** (CONST LOGFONT *lplf);

- LOGFONT 구조체가 지정하는 특성의 논리 폰트를 생성하는 함수

BOOL **ChooseFont** (LPCHOOSEFONT lpcf);

- 폰트 공용 대화상자를 여는 함수

폰트 선택하기

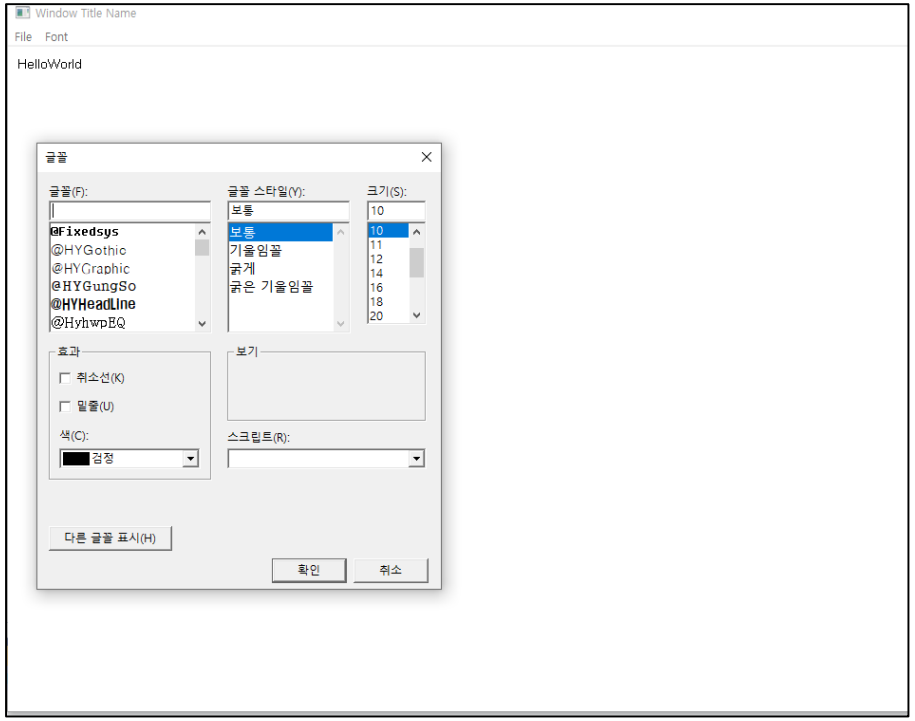
- 사용 예)

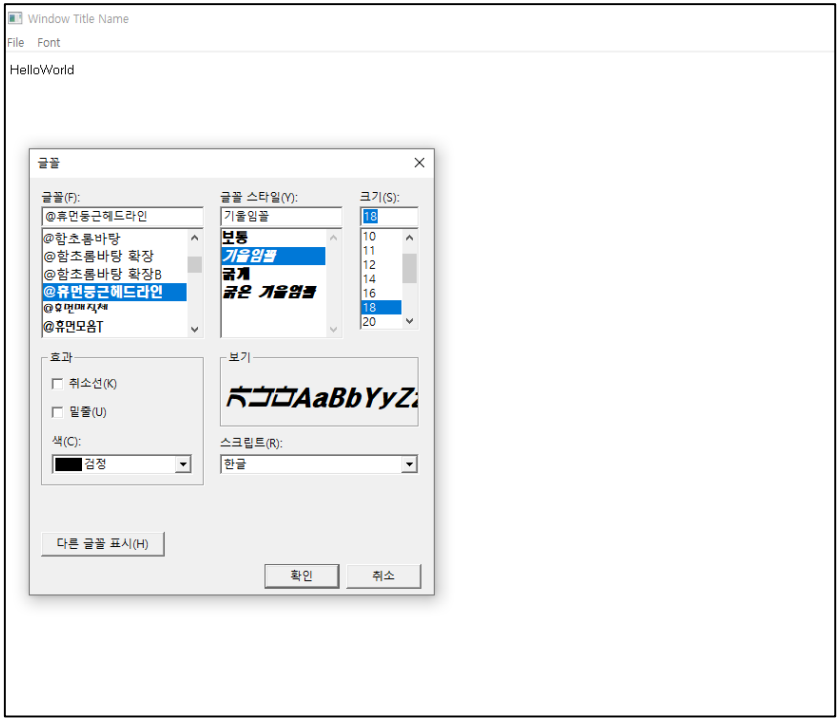
```
CHOOSEFONT      FONT;
HFONT           hFont, OldFont;
static LOGFONT  LogFont;
static COLORREF fColor;

 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


Window Title Name
File Font
HelloWorld
글꼴
글꼴(F):
글꼴 스타일(Y):
크기(S):
보통
10
@Fixedsys
@HYGothic
@HYGraphic
@HYGungSo
@HYHeadLine
@HyhwpEQ
보통
기울임꼴
굵게
굵은 기울임꼴
10
11
12
14
16
18
20
효과
취소선(K)
밑줄(U)
색(C):
검정
보기
스크립트(R):
다른 글꼴 표시(H)
확인
취소


Window Title Name
File Font
HelloWorld
글꼴
글꼴(F):
글꼴 스타일(Y):
크기(S):
@휴먼둥근헤드라인
기울임꼴
18
@함초롬바탕
@함초롬바탕 확장
@함초롬바탕 확장B
@휴먼둥근헤드라인
@휴먼매직체
@휴먼모음T
보통
기울임꼴
굵게
굵은 기울임크
10
11
12
14
16
18
20
효과
취소선(K)
밑줄(U)
색(C):
검정
보기
AaBbYyZz
스크립트(R):
한글
다른 글꼴 표시(H)
확인
취소


Window Title Name
File Font
HelloWorld

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(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** (LPCHOOSCOLOR color);

- 색상 공용 대화상자를 여는 함수

색상 선택하기

- CHOOSECOLOR 구조체

```
typedef CHOOSECOLOR {
        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_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;
```

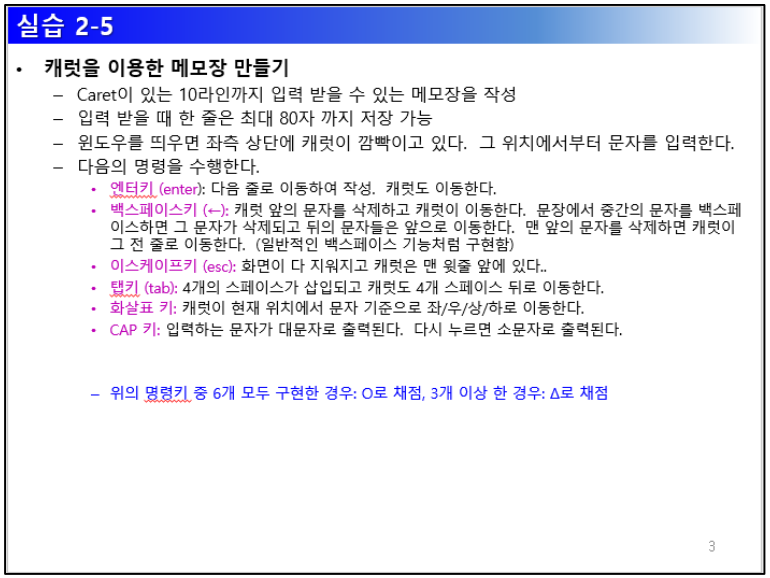
결과 화면

```
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;
```

실습 8-1

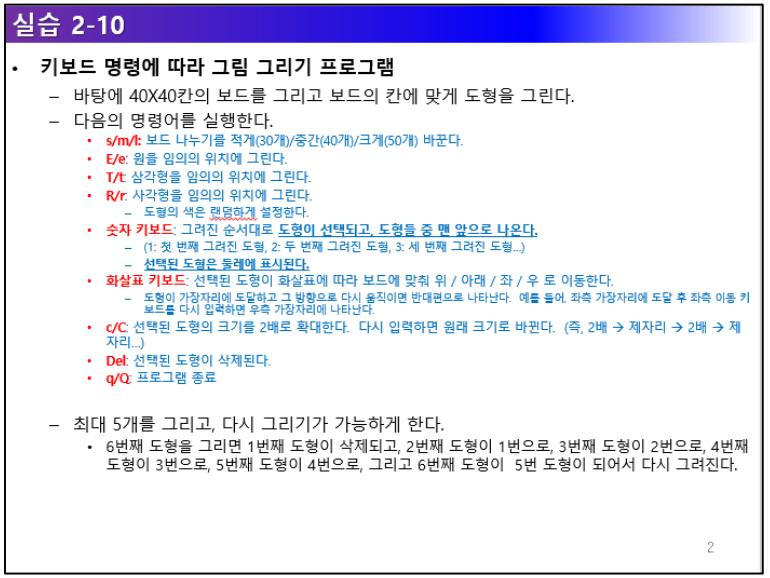
- **파일 입출력 기능이 있는 메모장 만들기**
  - 실습 2-5(메모장 만들기)에서 구현한 캐럿이 있는 10라인 메모장 실습에 파일 입출력 기능을 추가한다
  - 파일 공용 대화상자를 띄워서 입출력 할 파일을 선택하도록 한다.
  - 선택된 파일을 사용하여 텍스트를 읽기/쓰기를 한다.

  - 실습 2-5를 못 한 경우:
    - 8라인 정도의 파일을 만들기
    - 프로그램 시작했을 때 저장한 파일을 출력하면서 시작하기
    - 출력된 내용에 추가로 문자열 입력하기
    - 저장한 파일 수정하고 파일 공용 대화상자를 사용하여 파일 저장



실습 8-2

- **그래픽 데이터 파일에 저장하기**
  - 실습 2-10 (그림 그리기 프로그램)을 사용하여 화면에 그린 도형들을 저장하고 다시 읽을 수 있는 기능을 추가하도록 한다.
    - 저장할 내용:
      - 객체의 개수
      - 객체 마다
        - » 종류 (원, 삼각형, 사각형)
        - » 좌표 및 크기
        - » 색



  - 실습 2-10을 못한 경우
    - 파일에 데이터 미리 저장하기
    - 화면에 저장된 데이터를 읽어 객체 그리기
      - 프로그램 시작할 때 바로 그리기
      - 도형 종류: 원, 삼각형, 사각형
      - 도형 색상: 랜덤한 색상
    - 저장된 파일 수정한 후 위의 내용을 저장

3. PeekMessage () 함수

- 지금까지 우리가 사용했던 메시지루프

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

- GetMessage()함수는 메시지 큐에 대기중인 메시지가 없을 경우 메시지가 전달될 때까지 리턴하지 않고 무한히 대기한다.

- 특별한 일을 하지 않고 대기하는 시간에 다른 일을 하려면 이 함수 대신 PeekMessage()함수를 사용하는 것이 좋다.

PeekMessage () 함수

- 두 함수의 프로토타입
  - BOOL GetMessage (LPMSG lpMsg, HWND hWnd,
    UINT wMsgFilterMin, UINT wMsgFilterMax);

  - 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.
      - PM_NOYIELD: 다른 스레드로 제어를 양보하지 않는다.
    - 리턴값: 메시지 큐에 메시지가 있으면 0이 아닌 값을 리턴, 메시지가 없으면 0을 리턴

PeekMessage () 함수

- PeekMessage() 함수는
  - GetMessage()함수처럼 메시지 큐에서 메시지를 읽는다.
  - 이 함수는 GetMessage()함수와 달리 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()함수는 WM_QUIT메시지를 받으면 FALSE를 리턴하여 무한 메시지 루프를 빠져나올 수 있도록 하지만 PeekMessage()함수는 메시지 존재 여부만 알려주므로 무한 메시지 루프를 빠져 나올 수 없다.

PeekMessage () 함수

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

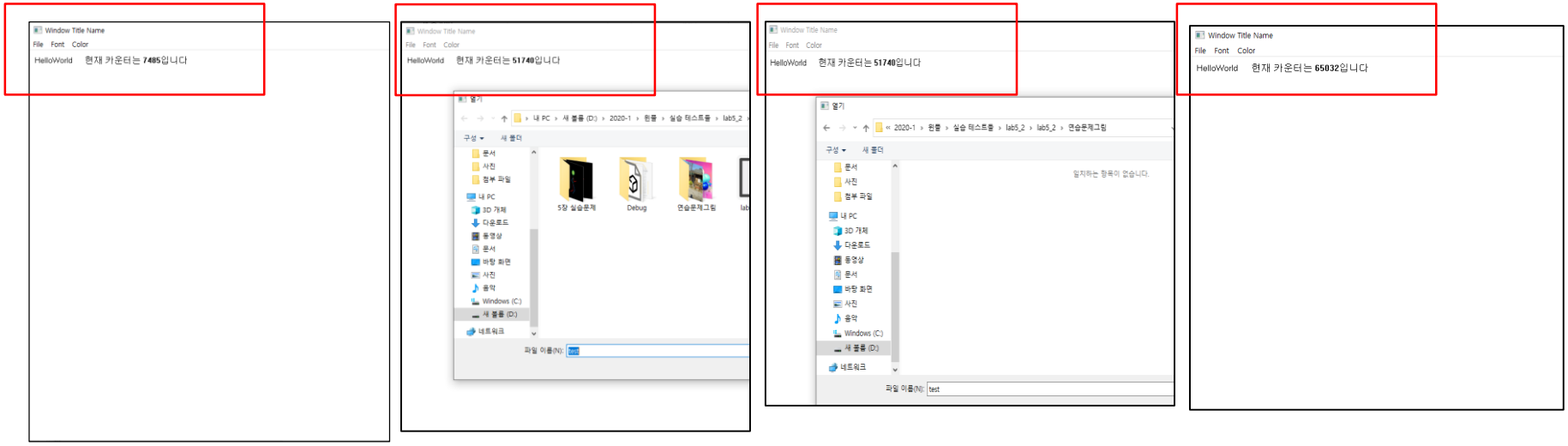
```
while (1){
    if ( PeekMessage (&msg, NULL, 0, 0, PM_REMOVE ) ){
        if ( msg.message == WM_QUIT )
                break;
        TranslateMessage (&msg);
        DispatchMessage (&msg);
       }
}
```

- 시간이 비교적 오래 걸리는 함수나 코드 부분을 실행할 때 함수나 코드내에 PeekMessage() 함수로 메시지 존재 여부를 판단하는 코드를 추가하여 사용자 입력 같은 이벤트에 즉각적으로 반응하여 처리할 수 있도록 해야 한다.

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

- SHORT **GetAsyncKeyState** (int vKey);
  - 현재 키 상태를 알아오는 함수로 키가 눌려졌을 때(down)나 떨어졌을 때(up) 호출됨
  - WM_KEYDOWN 메시지는 키가 눌러질 때 보내지는 메시지로, 키를 계속 누르고 있다는 것을 알려주지는 않는다.
    - 한 개의 키를 누른 상태에서 다른 키를 누르면, 두 번째 누른 키만 전달된다.
  - 키의 현재 상태, 즉 키가 눌러졌는지 아닌지를 조사해야 한다.
    - 키가 눌렸는가, 언제부터 눌렸는가를 알아낼 때 사용
  - GetAsyncKeyState는 메시지 처리 시점의 키 상태를 조사
  - 메시지 큐로 가지 않고 시스템에서 바로 상태 파악

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

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

- if (GetAsyncKeyState (KEYCODE) & 0x8000)
  1) 이전에 누른적이 없고, 호출 시점에 안 눌린 상태 ➔ 0x0000
     0x0000 & 0x8000 ➔ 0이 반환 ➔ false
  2) 이전에 누른 적이 없고, 호출 시점에 눌린 상태 ➔ 0x8000
     0x8000 & 0x8000 ➔ 0이 아니므로 ➔ true
  3) 이전에 누른 적이 있고, 호출 시점에 눌린 상태 ➔ 0x8001
     0x8001 & 0x8000 ➔ 0이 아니므로 ➔ true
  4) 이전에 누른 적이 있고, 호출 시점에 안 눌린 상태➔ 0x0001
     0x0001 & 0x8000 ➔ 0이 반환 ➔ false

GetAsyncKeyState () 함수

- 사용 예) 스페이스키의 현재 상태

```
case WM_KEYDOWN:
        if ( GetAsyncKeyState (VK_SPACE) & 0x8000)          //--- 정확한 시점에서 키눌림 상태를 체크
                //--- 스페이스키가 눌려짐
        else
                //--- 스페이스키가 안 눌려짐
```

- 사용 예) left 키와 컨트롤 키를 함께 누른 경우

```
case WM_KEYDOWN:
        if ( (wParam == VK_LEFT) && (GetAsyncKeyState (VK_CONTROL) & 0x8000) )
                                                                //--- Left 키와 control 키
                ComboAttack ();
        else                                                    //--- 안 눌려짐
                SingleBullet ();
```

GetAsyncKeyState () 함수

- SHORT GetKeyState (int vKey):
  - 해당 키가 눌려졌는지, 현재 토글 상태는 무엇인지 알아낼 때 사용하는 함수
    - 토글 상태란 키가 한번 누르면 불이 켜지고 다시 한번 누르면 불이 꺼지는 상태 (예로 Caps Lock이나 Num Lock 경우)
  - 리턴값:
    - 음수값: 해당 키가 눌린 상태
    - 음수가 아닐 경우: 해당 키가 눌리지 않은 상태
  - 사용 예)

```
case WM_KEYDOWN:
        if ( (wParam == VK_LEFT) && (GetKeyState(VK_CONTROL) < 0) )
            //--- 컨트롤 키와 왼쪽화살표 키가 눌림
```

- GetAsyncKeyState 와 GetKeyState 차이점
  - 메시지가 처리될 때(현재 키 상태)의 상황을 조사하는가: GetAsyncKeyState
    - 호출된 시점에서 키 상태를 조사, 메시지 큐를 거치지 않고 바로 리턴해주므로 키 입력을 바로 처리해줄 수 있다.
  - 메시지가 발생되었을 때의 키상태를 조사하는가: GetKeyState
    - 호출된 시점에서 메시지 큐를 거치며, 메시지 발생 후의 상태를 리턴하게 된다.
  - 현재 키보드의 상태값을 알아내길 원한다면 GetAsyncKeyState 사용하는 것을 추천

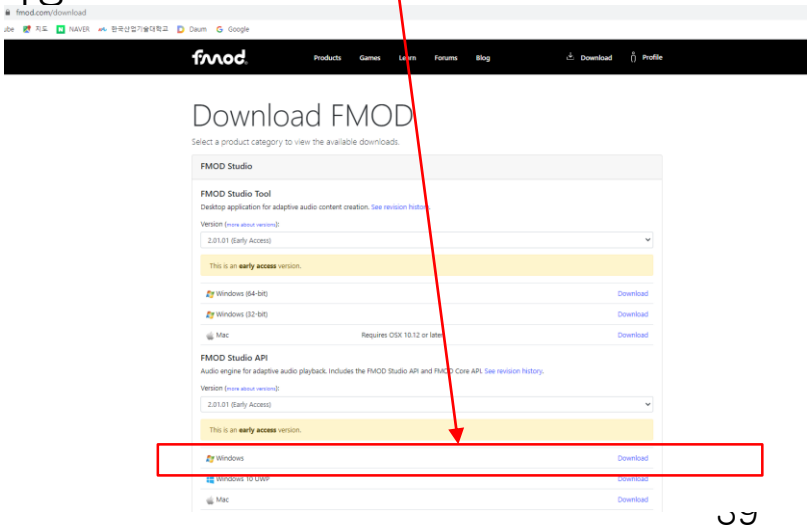
5. 사운드 이용하기: PlaySound 함수

- 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_RESORCE가 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

FMOD 사운드 시스템

- FMOD 사운드 엔진:
  - FireLight Technologies(www.fmod.com)에서 만든 음향 미들웨어
  - fmod.com에서 "**FMOD Studio ➔ FMOD Studio API ➔ Windows**" 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
        - » 링커 → 입력 → 추가종속성 → fmod_vc.lib 추가
      - **프로젝트 폴더에** fmod.dll (C:₩Program Files (x86)₩FMOD SoundSystem₩FMOD Studio API Windows₩api₩core₩lib₩x86₩fmod.dll) 저장
      - **헤더파일 추가하기**
        - » #include <fmod.h>
  - 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

- 사운드 시스템 생성과 초기화
  - 시스템 객체 선언 및 생성
  - Fmod 시스템 초기화하기

  - 사용 예)

```
#include <fmod.h>
FMOD_SYSTEM          *soundSystem;                          //--- FMOD system 변수 선언

FMOD_System_Create (&soundSystem);                          //--- FMOD system 객체 생성
FMOD_System_Init (soundSystem, 10, 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 (NULL로 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

- 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

- 사용 예) 명령어 p: 배경음악을 번갈아 재생, left&control: 이펙트 0번 출력, right&control: 이펙트 1번 출력, 화면에 사각형이 오른쪽으로 이동

```
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++)
	{
		wsprintfA (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]);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage,    WPARAM wParam, LPARAM lParam)
{
	HDC hdc;
	PAINTSTRUCT ps;
	static int x[2], y[2], sound, xDir;
	int i;

	switch (iMessage) {
		case WM_CREATE:
			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
			break;
```

동시 키 입력을 통한 이펙트 음악 재생하기 샘플 코드

```
        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;
                if (x[0] > nWidth) xDir *= -1;
                else if (x[0] < 0) xDir *= -1;
                InvalidateRect(hWnd, NULL, true);
                break;

        case WM_KEYDOWN:
                FMOD_Channel_Stop (Channel[1]);                                          //--- 현재 채널1에서 재생되는 사운드 정지
                if ((wParam == VK_LEFT) && (GetAsyncKeyState(VK_CONTROL) & 0x8000))      //--- control와 Left 키
                  FMOD_System_PlaySound (System, effectSound[0], NULL, 0, &Channel[1]); //--- 이펙트 0번을 1번 채널에서 생성
                else if ( (wParam == VK_RIGHT) && (GetKeyState(VK_CONTROL) < 0) )         //--- control와 Right 키
                  FMOD_System_PlaySound (System, effectSound[1], NULL, 0, &Channel[1]); //--- 이펙트 1번을 1번 채널에서 생성
                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], NULL, 0, &Channel[0]); //--- 배경 사운드를 0번 채널에서 재생
                                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, iMessage, wParam, lParam));
}
```

MCI (Media Control Interface)

- MCI
  - 사운드, 동영상, MIDI 등 여러가지 멀티미디어 요소에 대한 지원을 장치독립적으로 제공하는 일종의 라이브러리
  - 다양한 하드웨어 환경에서 실행될 수 있다

- 사용 방법
  - 명령 메시지 방식: MCI 디바이스에게 메시지를 보내 명령을 내림

- 라이브러리 링크
  - 라이브러리 winmm.lib 링크
    - 프로젝트 속성 → 링커 → 추가종속성에 winmm.lib 추가

- 용어
  - 장치 (Device): MCI의 연주 대상이 되는 모든 것

MCI (Media Control Interface)

- 객체 선언
  - MCI_OPEN_PARMS mciOpen; //파일을 로드
  - MCI_PLAY_PARMS mciPlay; //파일을 재생

- 구조체

```
typedef struct {
        DWORD_PTR    dwCallback;
        MCIDEVICEID wDeviceID;                  //--- 리턴되는 디바이스 id
        LPCTSTR      lpstrDeviceType;           //--- 파일 타입: MPEGVideo(mp3),  WAVEAudio(wav)
        LPCTSTR      lpstrElementName;          //--- 파일 이름
        LPCTSTR      lpstrAlias;
} MCI_OPEN_PARMS;

typedef struct {
        DWORD_PTR dwCallback;                   //--- MCI_NOTIFY 플래그를 사용할 윈도우 핸들
        DWORD      dwFrom;
        DWORD      dwTo;
} MCI_PLAY_PARMS;
```

MCI (Media Control Interface)

- 함수
  - MCIERROR **mciSendCommand** (MCIDEVICEID IDDevice, UINT uMsg, DWORD_PTR fdwCommand, DWORD_PTR dwParam);
    - 리턴값: 메시지가 성공했으면 0값이 리턴
    - IDDevice: MCI 장치 id, 메시지가 MCI_OPEN이면 NULL
    - uMsg: 메시지
      - MCI_OPEN: 디바이스를 초기화
      - MCI_PLAY: 데이터를 출력
      - MCI_PAUSE: 데이터를 멈춤
      - MCI_RESUME: 데이터를 다시 시작
      - MCI_SEEK: 위치 찾기
      - MCI_CLOSE: 디바이스 해제하기
    - fdwCommand: 메시지의 플래그
      - MCI_NOTIFY: 알림 플래그
        - » 연주가 종료될 때 지정한 윈도우로 MM_MCINOTIFY 통지메시지를 보낸다
      - MCI_WAIT: 대기 플래그
        - » 연주가 완전히 종료할 때까지 대기
    - dwParam: 메시지를 위한 값이 저장되어 있는 구조체 (MCI_OPEN_PARMS 구조체 또는 MCI_PLAY_PARMS 구조체 값)

MCI (Media Control Interface)

- 메시지:
  - MCI_OPEN: 디바이스를 초기화
  - MCI_PLAY: 데이터를 출력

mciSendCommand 함수에서

메시지	플래그	
MCI_OPEN	MCI_OPEN_TYPE	디바이스 타입을 lpstrDeviceType 멤버로 보냄
	MCI_OPEN_ELEMENT	파일 이름을 lpstrElementName 멤버로 보냄
MCI_PLAY	MCI_NOTIFY	알림 플래그
MCI_PAUSE	MCI_WAIT	디바이스를 멈춤
MCI_RESME	MCI_WAIT	멈춘 디바이스를 다시 시작

MCI (Media Control Interface)

- 발생 메시지
  - MM_MCINOTIFY: 동작 완료한 것을 통지
    - wParam: wFlags
      - MCI_NOTIFY_SUCCESSFUL: 콜백 함수 조건 만족한 상태
      - MCI_NOTIFY_FAILURE: 디바이스 에러 발생
      - MCI_NOTIFY_SUPERSEDED: 콜백 함수 개시에 필요한 조건 변경
      - MCI_NOTIFY_ABORTED: 커맨드 중단
    - lParam: lDevID

MCI (Media Control Interface)

- 디바이스 오픈
  - 메시지를 보내 디바이스 오픈

```
mciOpenParms.lpstrDeviceType = (LPCWSTR)(L"MPEGVideo");
mciOpenParms.lpstrElementName = (LPCWSTR)(lpszWave);
mciSendCommand (0, MCI_OPEN, MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                (DWORD)(LPVOID)&mciOpenParms);

wDeviceID = mciOpenParms.wDeviceID;          //--- id 저장하기
mciPlayParms.dwCallback = (DWORD)hWnd;       //--- 윈도우 핸들 저장하기
```

- 디바이스 연주하기
  - MCI_PLAY 메시지를 보내 디바이스 연주

```
mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY, (DWORD)(LPVOID)&mciPlayParms);
```

MCI (Media Control Interface)

- 사용 예) 왼쪽 마우스 버튼: 플레이 시작, 오른쪽 마우스 버튼:플레이 종료
  s: 잠시 멈춤, p: 다시 플레이,

```
#include "mmsystem.h"
UINT wDeviceID=0;

LRESULT CALLBACK WndProc (HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM lParam)
{
	HDC hdc;
	PAINTSTRUCT ps;
	DWORD Result;
	TCHAR str[256];
	TCHAR Mes[50]= L"왼쪽 버튼을 누르면 사운드를 연주합니다.";

	switch (iMessage) {
		case WM_LBUTTONDOWN:
			Result = PlaySoundFile (hWnd, (LPSTR)(L"Backmusic.mp3"));
			break;

		case WM_RBUTTONDOWN:
			if (wDeviceID)
				mciSendCommand (wDeviceID, MCI_CLOSE, 0, (DWORD)NULL);
			break;
```

MCI (Media Control Interface)

```
        case WM_CHAR:
                if ( wParam == 's')
                        mciSendCommand (wDeviceID, MCI_PAUSE, MCI_WAIT, (DWORD)(LPVOID)&mciPlayParms);
                else if ( wParam =='p')
                        mciSendCommand (wDeviceID, MCI_RESUME, MCI_WAIT, (DWORD)(LPVOID)&mciPlayParms);

        case MM_MCINOTIFY:
                switch (wParam) {
                case MCI_NOTIFY_SUCCESSFUL:
                case MCI_NOTIFY_FAILURE:
                case MCI_NOTIFY_SUPERSEDED:
                case MCI_NOTIFY_ABORTED:
                        mciSendCommand (wDeviceID, MCI_CLOSE, 0, (DWORD)NULL);
                break;
                }
                return 0;

        case WM_DESTROY:
                if (wDeviceID)
                        mciSendCommand(wDeviceID, MCI_CLOSE, 0, (DWORD)NULL);
                PostQuitMessage(0);
                return 0;
    }
    return(DefWindowProc(hWnd,iMessage,wParam,lParam));
}
```

MCI (Media Control Interface)

```
DWORD PlaySoundFile (HWND hWndNotify, LPSTR lpszWave)
{
	DWORD Result;
	MCI_OPEN_PARMS mciOpenParms;
	MCI_PLAY_PARMS mciPlayParms;

	//--- 장치를 Open: 디바이스 타입과 파일 이름을 전송
	//--- 장치의 ID를 발급받는다.
	mciOpenParms.lpstrDeviceType = (LPCWSTR) (L"MPECVideo");
	mciOpenParms.lpstrElementName = (LPCWSTR)(lpszWave);
	Result= mciSendCommand (0,MCI_OPEN, MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                                                (DWORD)(LPVOID) &mciOpenParms);
	if (Result) {
		return Result;
	}
	wDeviceID=mciOpenParms.wDeviceID;

	mciPlayParms.dwCallback=(DWORD) hWndNotify;
	Result = mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY, (DWORD)(LPVOID)&mciPlayParms);
	if (Result) {
		mciSendCommand(wDeviceID, MCI_CLOSE, 0, (DWORD)NULL);
		return Result;
	}
	return 0;
}
```

오늘은

- 파일 입출력
  - 실습 2개

- 공용 대화상자
- 유용한 함수들
  - 사운드
  - 동시 키입력

- 다음 시간에는
  - 맵툴 소개

- 다음주에는
  - 수업 진도 없음
  - 15주차의 수업 소개: 시험과 프로젝트에 대하여

- 다음시간에 만나요~