

그 외 알아두면 좋을 내용들

2024년도 1학기 윈도우 프로그래밍

1. 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 있다.
 - 백그라운드(background)로 작업을 처리해야할 때 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 나을 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

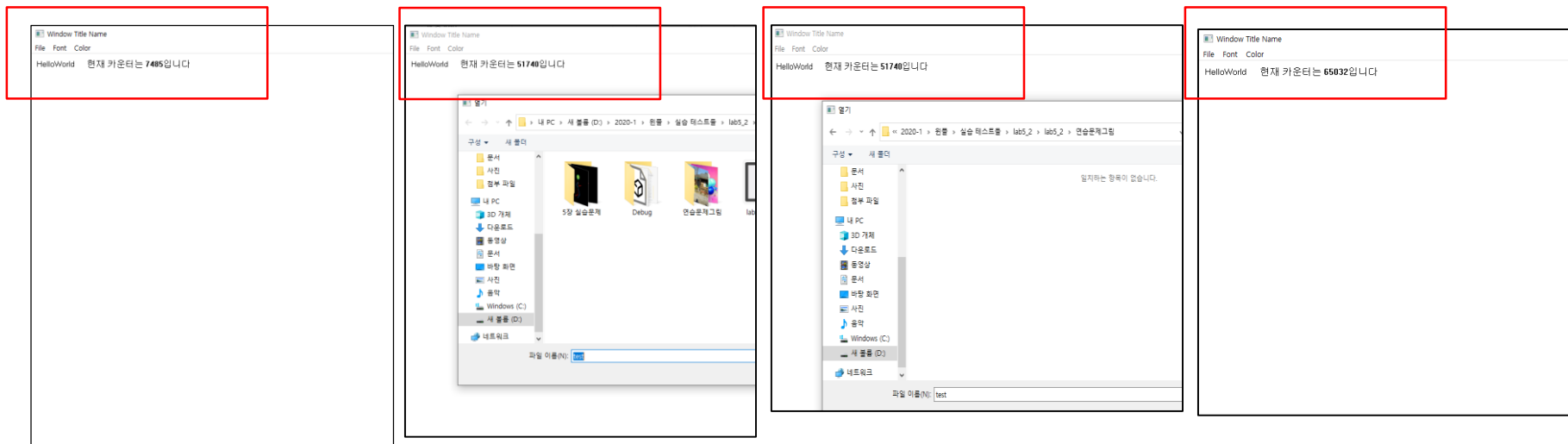
- PeekMessage()함수를 사용한 메시지 루프는 다음과 같이 구현한다.

```
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));
    }
}
```



2. 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

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)

- space 키 현재 상태 체크

```
case WM_KEYDOWN:
```

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

```
        //-- space 키가 눌려짐
```

```
    Else
```

```
        //-- space 키가 안 눌려짐
```

- 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) ) {           //--- 컨트롤 키와 왼쪽 화살표 키 상태 체크  
        //--- 컨트롤 키와 왼쪽 화살표 키가 눌렸음  
    }  
    else {  
        //--- 두 키가 동시에 눌러지지 않았음  
    }
```

GetAsyncKeyState () 함수 / GetKeyState () 함수

- GetAsyncKeyState 와 GetKeyState 차이점
 - **GetAsyncKeyState 함수: “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 리턴 해주므로 키 입력을 바로 처리해줄 수 있다.
 - **GetKeyState 함수: “키가 눌렸는가 ” 와 “키의 토글 상태가 무엇인가 ” 를 알아낼 때 사용**
 - 토글 상태란: Caps Lock, Num Lock 키가 한번 누르면 불이 켜지고, 다시 누르면 불이 꺼지는 상태 등
 - 현재 키보드의 상태값을 알아내길 원한다면 GetAsyncKeyState 사용하는 것을 추천

3. 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

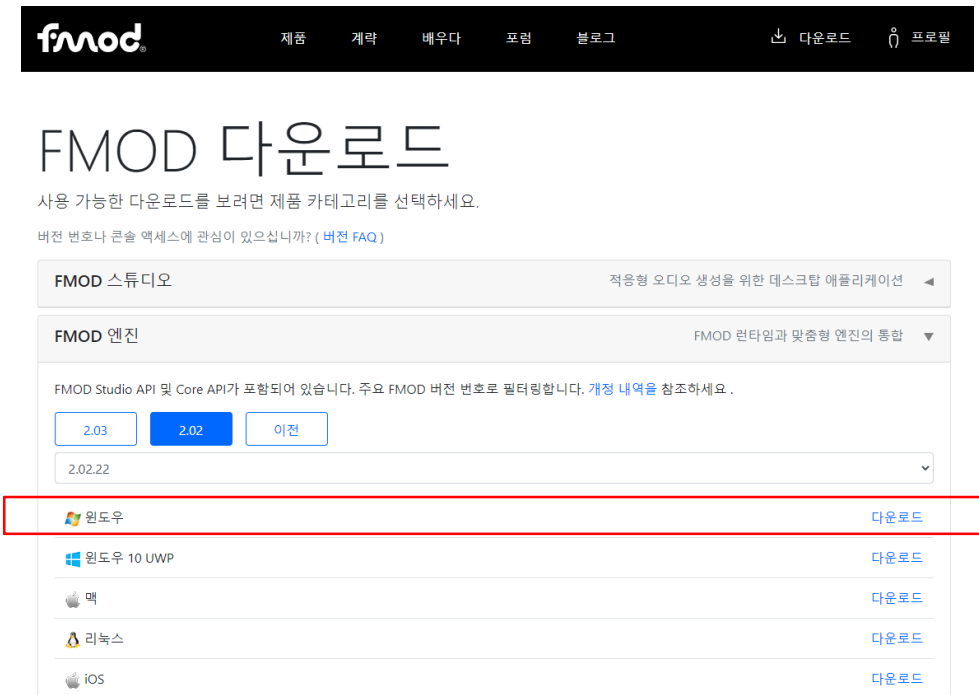
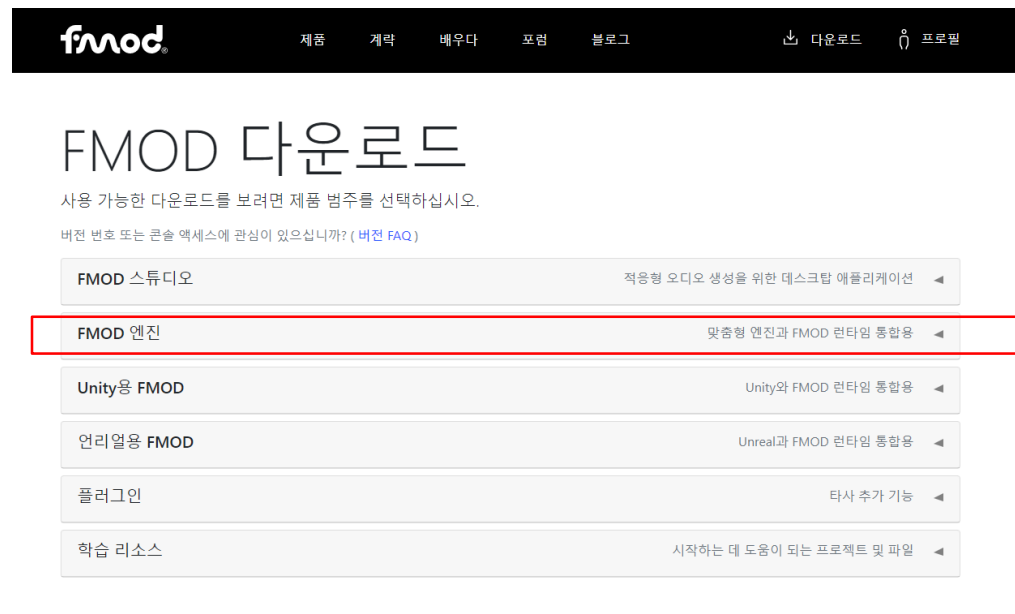
BOOL **PlaySound** (LPCSTR pszSound, HMODULE hmode, DWORD fdwSound);

- 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

4. 사운드 이용하기: FMOD 사운드 시스템

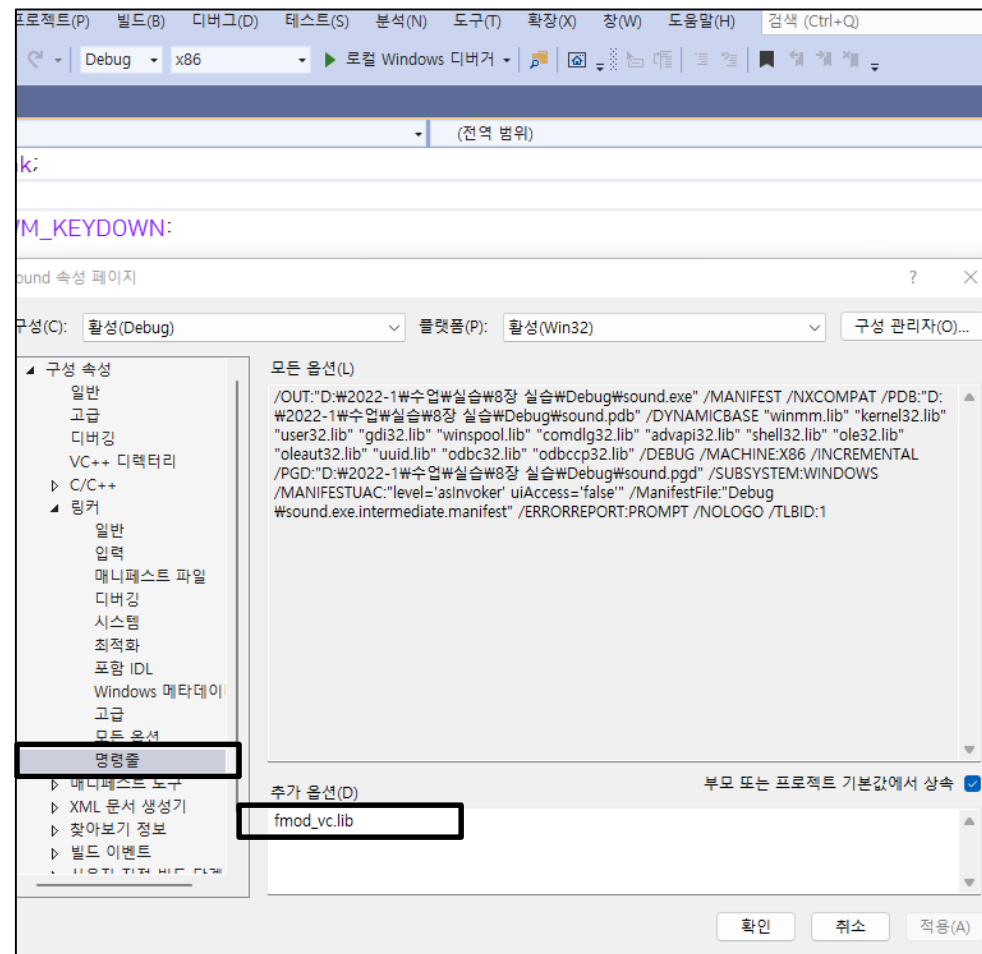
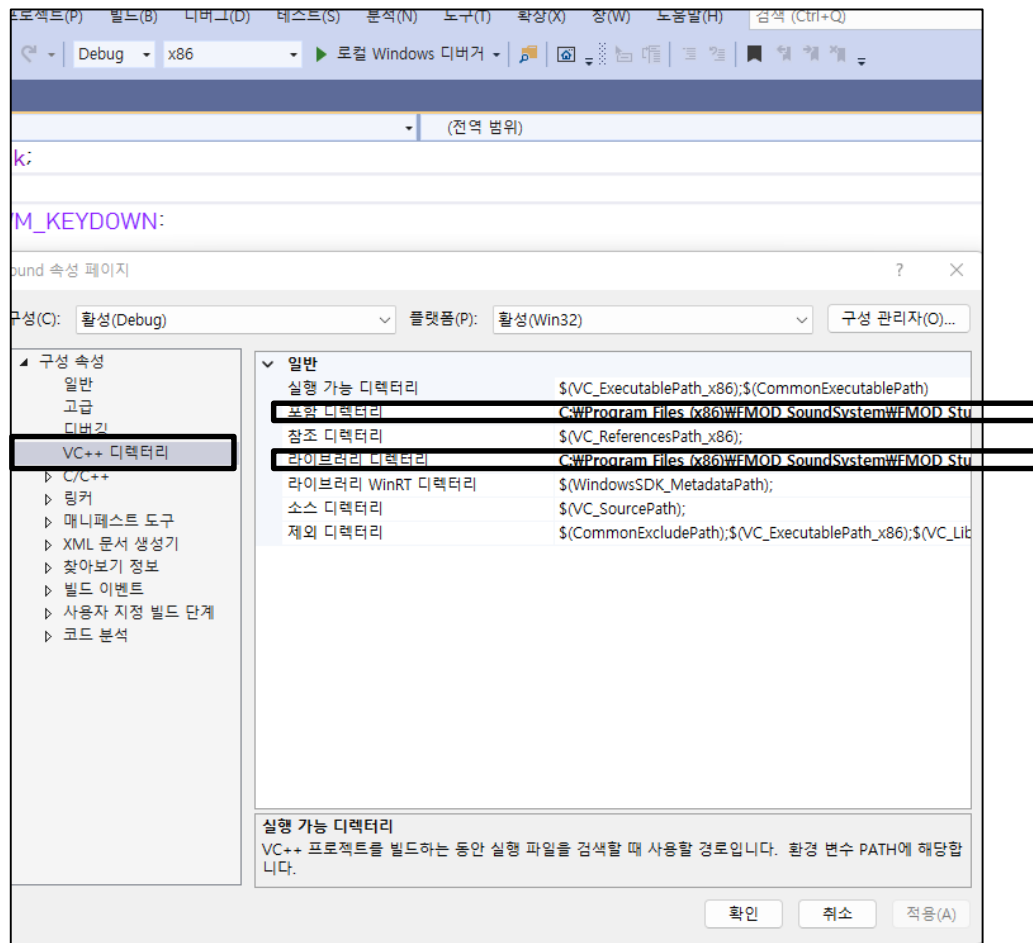
- FMOD 사운드 엔진:
 - FireLight Technologies(fmod.com)에서 만든 음향 미들웨어
 - 출력 가능한 사운드 형식
 - AIFF, ASF, ASX, DLS, FLAC, FSB, IT, M3U, MID, MOD, MP2, MP3, OGG, PLS, RAW, S3M, VAG, WAV, WMA, XM, XMA 등 대부분의 사운드 형식 사용 가능
 - fmod.com에서 "download → FMOD Engine → Windows (2.** 버전)" 다운로드



4. 사운드 이용하기: FMOD 사운드 시스템

- FMOD 사운드 엔진:
 - 설치
 - 프로젝트에 설정하기
 - 프로젝트 속성에서 라이브러리와 헤더파일 포함하기
 - 1) VC++ 디렉토리 → 포함 디렉터리에
C:\Program Files (x86)\FMOD SoundSystem\FMOD Studio API Windows\Wapi\Wcore\Winc
 - 2) VC++ 디렉토리 → 라이브러리 디렉터리에
C:\Program Files (x86)\FMOD SoundSystem\FMOD Studio API Windows\Wapi\Wcore\Wlib\Wx64
 - 3) 라이브러리 추가하기
프로젝트 속성 → 링커 → 입력 → 추가종속성 → fmod_vc.lib 추가
또는, 프로젝트 속성 → 링커 → 명령줄 → fmod_vc.lib 추가
또는, 메인 코드에 #pragma comment (lib, " fmod_vc.lib ") 추가
 - 헤더파일 추가하기
 - » #include "fmod.hpp"
 - » #include "fmod_errors.h"
 - 프로젝트가 있는 폴더에 fmod.dll 추가하기
 - » C:\Program Files (x86)\FMOD SoundSystem\FMOD Studio API Windows\Wapi\Wcore\Wlib\Wx64\fmmod.dll 파일을 프로젝트 (*.vcxproj)가 있는 폴더에 복사하기

FMOD 사운드 시스템



FMOD 사운드 시스템

```
#include "fmod.hpp"
#include "fmod_errors.h"
```

```
FMOD::System* ssystem;
FMOD::Sound* sound1, * sound2;
FMOD::Channel* channel = 0;
FMOD_RESULT result;
void* extradriverdata = 0;
```

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

```
{
    switch (iMsg)
    {
        case WM_CREATE:
            result = FMOD::System_Create(&:ssystem);                //--- 사운드 시스템 생성
            if (result != FMOD_OK)
                exit(0);

            ssystem->init(32, FMOD_INIT_NORMAL, extradriverdata);    //--- 사운드 시스템 초기화
            ssystem->createSound ( "sound.mp3", FMOD_LOOP_NORMAL, 0, &sound1);    //--- 1번 사운드 생성 및 설정
            ssystem->createSound ( "sound.mp3", FMOD_LOOP_NORMAL, 0, &sound2);    //--- 2번 사운드 생성 및 설정

            break;
        case WM_CHAR:
            if (wParam == '1') {
                channel->stop ();
                ssystem->playSound(sound1, 0, false, &channel);        //--- 1번 사운드 재생
            }
            else if (wParam == '2') {
                channel->stop ();
                ssystem->playSound(sound2, 0, false, &channel);        //--- 2번 사운드 재생
            }
            else if (wParam == '3')
                channel->stop ();

            break;
        ...
    }
}
```

예제) 사운드 2개를 생성한 후,
키보드 입력에 따라 생성하기

FMOD 사운드 시스템

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

```
FMOD::System* ssystem;  
FMOD::Sound* sound1, * sound2;  
FMOD::Channel* channel = 0;  
FMOD_RESULT      result;  
void* extradriverdata = 0;
```

```
FMOD_RESULT SYSTEM::System_Create (SYSTEM **system);  
– FMOD 사운드 시스템을 생성한다  
• system: FMOD 시스템
```

```
FMOD_RESULT SYSTEM::init (int MaxChannels, FMOD_INITFLAGS flags, void *extraDriverdata);  
– FMOD 시스템 초기화  
• 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

```
FMOD_RESULT SYSTEM::createSound (const char *name_or_data, FMOD_MODE mode,  
    FMOD_CREATESOUNDEXINFO *exinfo, 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 설정)
 - sound: 새로 만든 sound 객체

FMOD 사운드 시스템

- 사운드 재생
 - 사운드 객체 (sound) 를 통해 읽혀진 사운드는 채널(channel)을 통해 재생

```
FMOD_RESULT SYSTEM::playSound (Sound *SoundFile, CHANNELGROUP *ChannelGroup, bool paused, 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 ChannelControl::SetVolume (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 Channel::stop (CHANNEL *channel)  
    - 시스템 닫기  
        • channel: 정지할 채널
```

- FMOD 해제하기
 - 사용 후 시스템을 해제한다.

```
FMOD_RESULT System::close ()  
    - 시스템 닫기
```

```
FMOD_RESULT System::release ()  
    - FMOD sound 객체 해제
```

- 유용한 함수들
 - PeekMessage 함수
 - 동시 키 입력
 - 사운드