윈도우 프로그래밍 기초

HCI Programming 2 (321190)
2008년 가을학기
9/16/2008
박경신

Overview

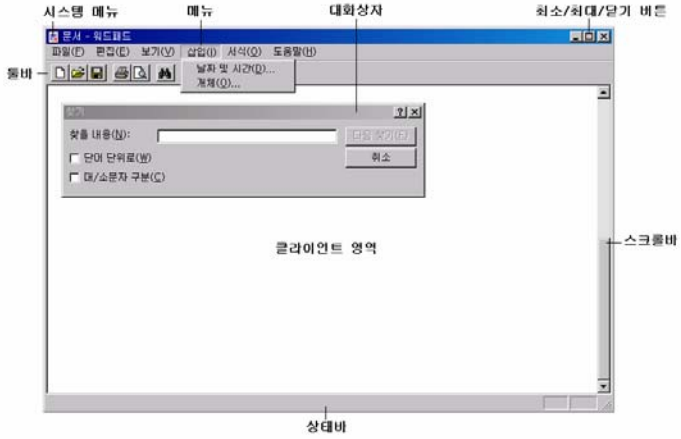
- 윈도우 운영체제와 윈도우 응용 프로그램의 특징 이해
- SDK 응용 프로그램 작성 과정, 기본 구조, 동작 원리 이해
- MFC 응용 프로그램 작성 과정, 기본 구조, 동작 원리 이해

Windows .NET

- 구조적 프로그램에서 객체지향 프로그램으로
  - SDK(Software Development Kit): 운영체제가 응용프로그램에 제공하는 서비스로 C언어 함수의 집합형태
  - MFC(Microsoft Foundation Class): SDK를 객체지향적으로 포장
  - ATL(Active Template Library): 분산환경 컴포넌트 개발

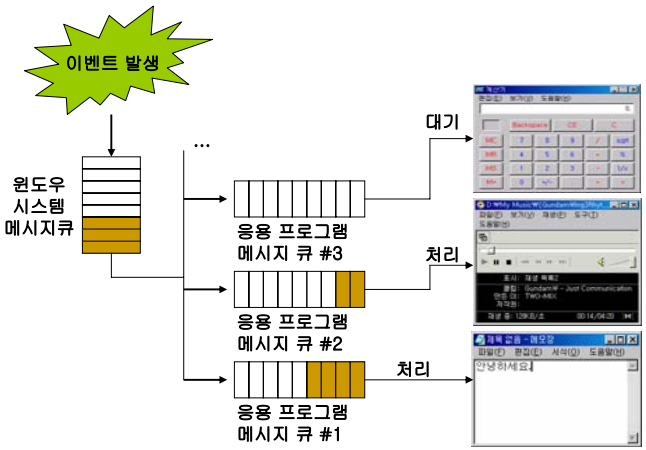
윈도우 운영체제 특징

- 그래픽 사용자 인터페이스 (Graphical User Interface, GUI) – 일관성 있는 사용자 인터페이스



윈도우 운영체제 특징

- 메시지 구동 구조 (Message-driven Architecture)



윈도우 운영체제 특징

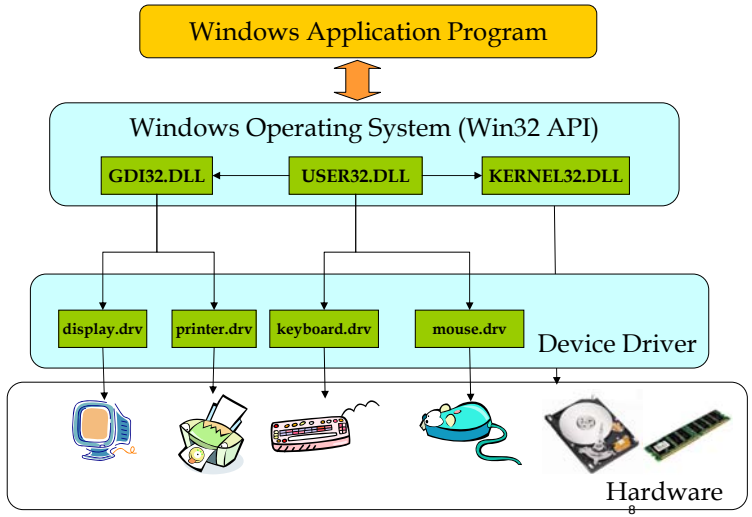
- 멀티태스킹 (Multi-Tasking)
  - 하나의 윈도즈 시스템에서 여러 개의 응용 프로그램을 수행
  - 응용 프로그램 사이의 상호작용 가능
- 멀티스레딩 (Multi-Threading)
  - 하나의 응용 프로그램에 여러 개의 실행 흐름을 생성
- 장치에 독립적
  - 장치 드라이버 (Device Driver)에 의해 주변 장치들을 제어

윈도우 운영체제

- 윈도우 시스템 구성 모듈

구분	Win32 API	역할
커널 모듈	KERNEL32.DLL	윈도우 시스템의 주요부분으로 메모리 관리, 파일입출력, 프로그램의 로드와 실행 등 운영체제의 기본 기능 수행
GDI 모듈	GDI32.DLL	화면이나 프린터와 같은 장치의 출력을 관리. (펜, 브러쉬, 폰트, 비트맵, 팔레트 등 관리)
사용자 인터페이스 모듈	USER32.DLL	윈도우, 다이얼로그, 메뉴, 커서, 아이콘 등과 같은 사용자 인터페이스 객체들을 관리

윈도우 운영체제 & 윈도우 응용 프로그램



윈도우 응용 프로그램 특징

- API 호출문 집합
  - API(Application Programming Interface) - 윈도우 운영체제가 응용 프로그램을 위해 제공하는 각종 함수 집합

응용 프로그램

```
call API#1
call API#2
...
call API#3
call API#4
...
call API#n
```


윈도우 응용 프로그램 특징

- 메시지 핸들러 집합
  - 메시지 핸들러 (Message Handler) - 메시지를 받았을 때 동작을 결정하는 코드
    - 키보드, 마우스, 메뉴, 등등
  - 윈도우 프로시저 (Window Procedure) – 이러한 메시지 핸들러의 집합

응용 프로그램

```
...
메시지 핸들러 #1
메시지 핸들러 #2
메시지 핸들러 #3
메시지 핸들러 #4
메시지 핸들러 #5
메시지 핸들러 #6
...
```

} 윈도우 프로시저: 메시지 핸들러 집합

윈도우 응용 프로그램 특징

- 실행 파일과 DLL 집합
  - DLL(Dynamic-Link Library): 프로그램이 실행 중에 호출할 수 있는 함수(코드)와 리소스

응용 프로그램

```
실행 파일
DLL #1
DLL #2
DLL #3
DLL #4
DLL #5
...
```


윈도우 응용 프로그램 특징

- 장치 독립적
  - 윈도우 시스템의 API를 사용하여 간접적으로 주변장치들을 제어

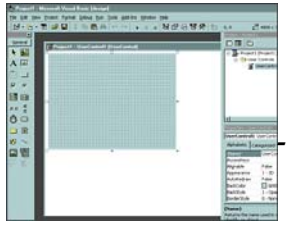
응용 프로그램 ↔ API ↔ 장치 드라이버 ↔ 주변 장치

윈도우 응용 프로그램 개발 도구

- SDK (Software Development Kit)
  - C 언어로 윈도우 API를 직접 호출해서 프로그램을 구현
  - 예, WIN32 API
- 장점
  - API를 직접 다루기 때문에 세부적인 제어가 가능하고, 윈도우 운영체제가 제공하는 모든 기능을 사용할 수 있다.
  - 생성 코드의 크기가 작고 속도가 빠르다.
- 단점
  - 생산성이 낮다.

윈도우 응용 프로그램 개발 도구

- RAD (Rapid Application Development)
  - 시각적으로 화면을 디자인하고 코드를 추가하는 방법으로 프로그램을 빠르게 개발
  - 예, Visual Basic, Delphi
- 장점
  - 생산성이 높다.
- 단점
  - 일반적으로 생성 코드의 크기가 크고 실행 속도가 느리다.
  - 운영체제가 제공하는 모든 기능을 활용한 세부적인 제어가 어렵다.

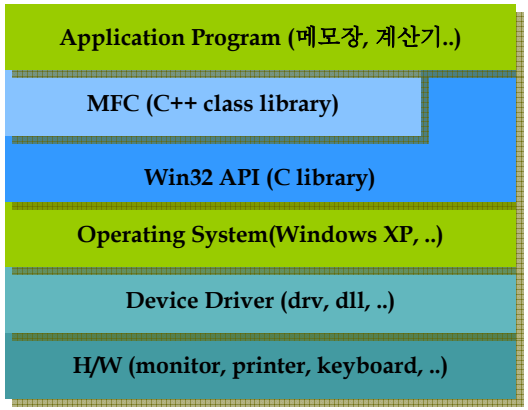


RAD 예
- Visual Basic

윈도우 응용 프로그램 개발 도구

- 클래스 라이브러리 (Class Library)
  - 윈도우 응용 프로그램 개발에 필수적인 기능을 객체 지향 언어를 이용하여 클래스로 제공
  - 예, MFC (Microsoft Foundation Class Library), 볼랜드 OWL (Object Windows Library)
- 장점
  - SDK보다 생산성이 높다.
  - RAD보다 생성 코드의 크기가 작고 실행 속도가 빠르다.
- 단점
  - 초기 학습에 필요한 기간이 길다.
    - 객체 지향 언어, C++
    - 클래스 라이브러리 구조와 기능, 각 클래스의 관계 파악

윈도우 응용 프로그램, 윈도우 시스템, 도구



Win32 SDK 윈도우 프로그램 구조

- WinMain 함수
  - 윈도우 클래스 구조 정의 및 등록
    - 윈도우 클래스 (WNDCLASS) - 윈도우 기본틀에 대한 정보(배경화면, 커서모양, 아이콘, 메뉴 등)을 정의하기 위한 구조체
  - 메인 윈도우 생성 및 출력
    - 윈도우 클래스로부터 인스턴스 생성
  - 메시지 루프
    - 윈도우 시스템으로부터 들어오는 메시지를 분석하고 처리
    - 메시지 큐에서 메시지를 하나씩 꺼내어 처리하기 위한 반복문
    - 응용 프로그램이 종료될 때까지 반복
- WinProc 함수
  - 윈도우시스템으로부터 받은 메시지를 처리하는 함수
  - 윈도우 시스템에서 호출하는 함수(call back)

Event & Message

- Event
  - 사용자의 물리적 조작에 의해 발생 (마우스 클릭, 키보드입력)
  - 윈도우 시스템, 응용프로그램 자체에서 발생
- Message
  - 이벤트 발생시 윈도우 시스템이 감지하여 메시지로 변환
  - 윈도우 시스템은 해당 응용프로그램으로 메시지 전달
  - 메시지는 각 이벤트에 대한 표준화된 상수값
  - winuser.h에 정의

```
#define WM_MOUSEFIRST                   0x0200
#define WM_MOUSEMOVE                    0x0200
#define WM_LBUTTONDOWN                  0x0201
#define WM_LBUTTONUP                    0x0202
#define WM_LBUTTONDBLCLK                0x0203
#define WM_RBUTTONDOWN                  0x0204
#define WM_RBUTTONUP                    0x0205
#define WM_RBUTTONDBLCLK                0x0206
#define WM_MBUTTONDOWN                  0x0207
#define WM_MBUTTONUP                    0x0208
#define WM_MBUTTONDBLCLK                0x0209
```

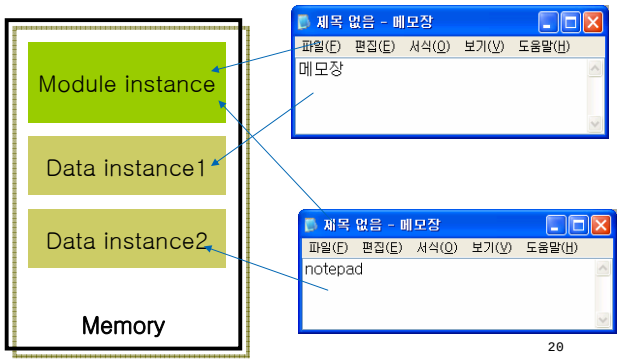

Handle

- 사용중인 객체를 식별하는 고유번호(32비트 정수형)
- 윈도우 시스템에서 부여

데이터 타입	의미
HINSTANCE	프로그램에 대한 핸들
HWND	윈도우에 대한 핸들
HDC	디바이스 컨텍스트에 대한 핸들
HCURSOR	커서에 대한 핸들
HICON	아이콘에 대한 핸들
HMENU	메뉴에 대한 핸들

Instance

- 인스턴스는 실제 메모리 상에 할당된 객체
- 윈도우 프로그램에는 코드 영역에 대한 모듈 인스턴스와 데이터 영역에 대한 데이터 인스턴스 존재



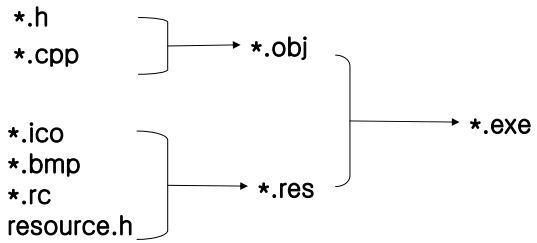
Device Context

- 출력에 필요한 정보를 위한 구조체
  - 출력 정보 : 폰트, 색상, 무늬, 굵기, 출력방법 등
- 출력장치를 제어하는 GDI (Graphic Device Interface) 함수 호출 시 사용

```
//DC 사용 예
HDC hdc= getDC(hWnd);  //현재 윈도우로부터 DC 얻어옴
//각종 출력문
TextOut(hdc, 100,100, “Beautiful world”, 20); // 문자열 출력
Rectangle(hdc, 30,40,60,70); //사각형 그리기
LineTo(hdc, 0, 200);  //라인 그리기
ReleaseDC(hWnd, hdc)  //DC 해제
```

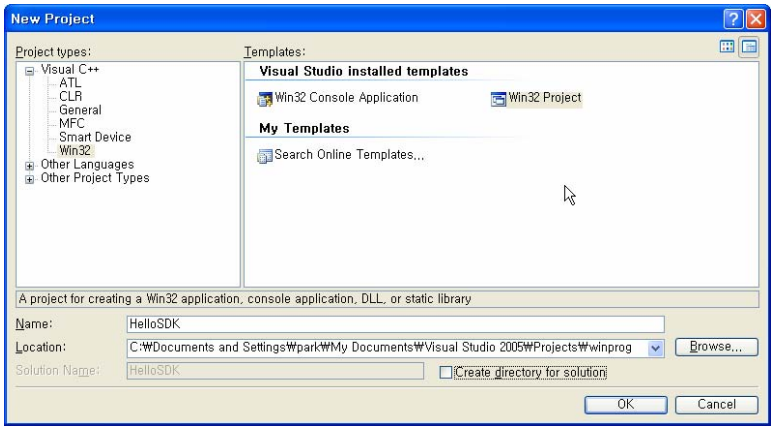

Resource

- 사용자 인터페이스를 구성하는 자원들의 정적 데이터
- 메뉴, 아이콘, 커서, 다이얼로그 등
- 리소스 스크립트 (Resource Script)에 의해 정의(.RC)
- 리소스는 프로그램 코드와 분리하여 작성되며 자체 컴파일(RC.exe) 과정으로 생성된 리소스 파일(.RES)은 링크시 통합



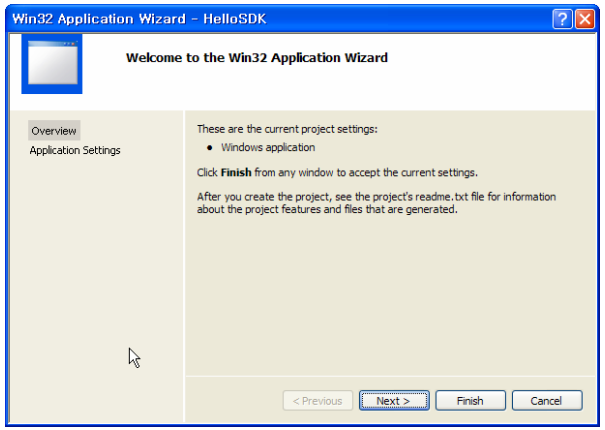
HelloSDK 예제 작성

- 프로젝트 생성



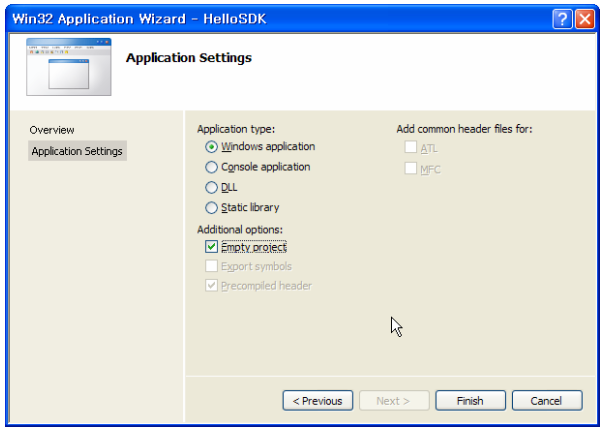
HelloSDK 예제 작성

- 1단계 옵션 설정



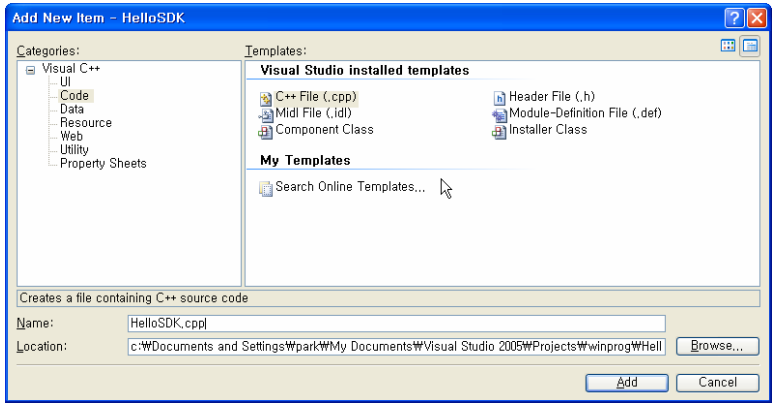
HelloSDK 예제 작성

- 1단계 옵션 설정



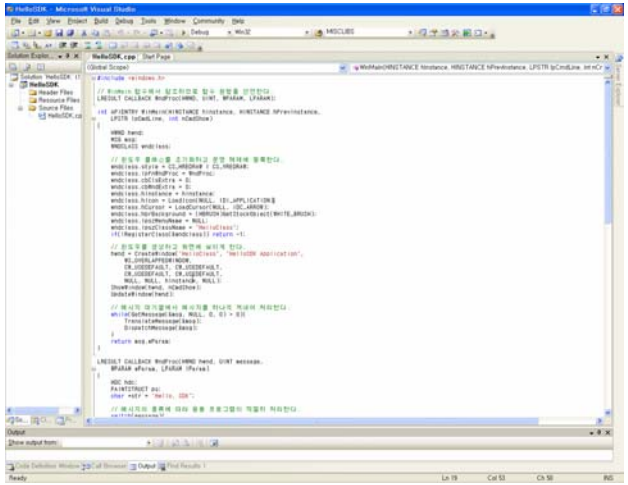
HelloSDK 예제 작성

- 소스 파일 추가



HelloSDK 예제 작성

- 코드 입력



HelloSDK 예제 작성

- 코드 입력

```
#include <windows.h>

// WinMain( ) 함수에서 참조하므로 함수 원형을 선언한다.
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    // 윈도우 클래스를 초기화하고 운영체제에 등록한다.
    wndclass.style = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = WndProc;
    wndclass.cbClsExtra = 0;
    wndclass.cbWndExtra = 0;
    wndclass.hInstance = hInstance;
```

HelloSDK 예제 작성

```
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = "HelloClass";
if(!RegisterClass(&wndclass)) return -1;

// 윈도우를 생성하고 화면에 보이게 한다.
hwnd = CreateWindow("HelloClass", "HelloSDK Application",
   WS_OVERLAPPEDWINDOW,
   CW_USEDEFAULT, CW_USEDEFAULT,
   CW_USEDEFAULT, CW_USEDEFAULT,
   NULL, NULL, hInstance, NULL);
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
```


HelloSDK 예제 작성

```
   // 메시지 대기열에서 메시지를 하나씩 꺼내 처리한다.
   while(GetMessage(&msg, NULL, 0, 0) > 0){
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
   HDC hdc;
   PAINTSTRUCT ps;
   char *str = "Hello, SDK";
```


HelloSDK 예제 작성

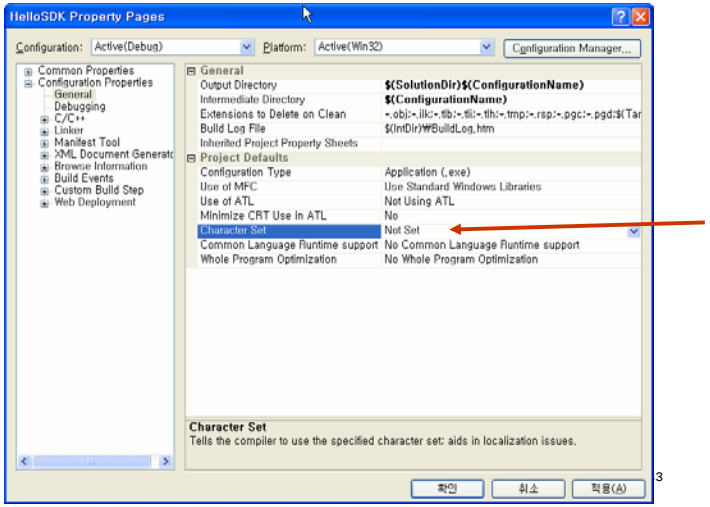
```
// 메시지 종류에 따라 응용 프로그램이 적절히 처리한다.
switch(message) {
case WM_CREATE:
   return 0;
case WM_PAINT:
   hdc = BeginPaint(hwnd, &ps);
   TextOut(hdc, 100, 100, str, lstrlen(str));
   EndPaint(hwnd, &ps);
   return 0;
case WM_LBUTTONDOWN:
   MessageBox(hwnd, "마우스를 클릭했습니다.",
      "마우스 메시지", MB_OK);
   return 0;
```


HelloSDK 예제 작성

```
   case WM_DESTROY:
      PostQuitMessage(0);
      return 0;
   }

   // 응용 프로그램이 처리하지 않으면 윈도우 운영체제가 처리한다.
   return DefWindowProc(hwnd, message, wParam, lParam);
}
```

HelloSDK 예제 작성

- 프로젝트 설정 변경



HelloSDK 예제

- F7 (Build Solution) -> F5 (Start Debugging)
- 실행 화면

HelloSDK 예제 분석

- 헤더 파일
  - 윈도우 API 함수의 원형
  - Data type, 구조체, MACRO 상수들이 선언

```
#include <windows.h>
```


HelloSDK 예제 분석

- 메인 함수 WinMain
  - WinMain() 프로그램을 시작하는 시작점
  - “windef.h”안에 #define APIENTRY WINAPI라고 지정되어 있음

```
int APIENTRY WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    ...
```

HelloSDK 예제 분석

- 메인 함수 WinMain
  - HINSTANCE는 Process ID (즉, Instance Handle)
  - hInstance
    - Win32 프로그램이 실행될 때 시작되는 주소값을 가지고 있는 포인터 변수
    - 리소스를 로드하는 함수들이 이 주소값을 참조해서 리소스를 참조
    - 메모장을 2개 실행하는 예 – 같은 프로그램이지만 할당되는 hInstance는 다름. 각각 고유의 값인 hInstance로 구분
  - hPrevInstance
    - 16-bit 윈도우의 잔재로 사용하지 않음. NULL 처리

```
int APIENTRY WinMain(HINSTANCE hInstance,
   HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
   ...
```


HelloSDK 예제 분석

- 메인 함수 WinMain
  - LPSTR는 32-bit pointer to a character string (즉, char *)
  - lpComLine
    - Command Line 명령행 인자를 담고 있는 문자열
    - "HelloSDK TEST1.TXT TEST2.TXT"처럼 프로그램을 실행하면, lpComLine은 "TEST1.TXT TEST2.TXT" 값을 가짐
  - nCmdShow
    - 프로그램이 시작될 때 윈도우 모양 (최대화면, 아이콘 표시 등) 결정
    - 윈도우의 최대, 최소, 정상상태로 실행
    - ShowWindow() 함수 호출시 사용

```
int APIENTRY WinMain(HINSTANCE hInstance,
   HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
   ...
```


HelloSDK 예제 분석

- 메인 함수 WinMain
  - HWND
    - Win32 프로그램 안에서 생성되거나 사용되는 모든 윈도우를 구별하거나 사용하는데 쓰임
    - hInstance가 프로그램에 하나씩 할당하는 반면, HWND는 프로그램 안의 윈도우를 다루기 위하여 다수의 HWND가 존재
  - WNDCLASS
    - 윈도우를 생성하는데 필요한 다양한 정보를 담고 있는 구조체
    - CreateWindow() 함수를 사용하기 전에 윈도우 클래스를 커널 (kernel)에 등록

```
int APIENTRY WinMain(HINSTANCE hInstance,
   HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    HWND hwnd;
    MSG msg;
    WNDCLASS wndclass;
```


HelloSDK 예제 분석

- 윈도우 클래스 초기화와 등록 WNDCLASS
  - GetStockObject() 이용
    - StockObject은 윈도우가 기본적으로 제공해주는 GDI (Graphic Device Interface) object. 주로 브러쉬와 펜이 사용됨.

```
wndclass.style = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc = WndProc; // 윈도우 프로시저 함수 주소 정의
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInstance;
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = "HelloClass";
if(!RegisterClass(&wndclass)) return -1;
```

HelloSDK 예제 분석

- 윈도우 생성 CreateWindow
  - 윈도우 핸들 = CreateWindow("지정된 클래스 이름", "윈도우 캡션 바에 표시되는 문자열", 윈도우 스타일, 윈도우 수평위치, 윈도우 수직위치, 윈도우 폭, 윈도우 높이, 부모 윈도의 핸들, 메뉴 핸들, 인스턴스 핸들, WM_CREATE 메시지가 윈도우 프로시저에 보내질 때의 parameter);

```
hwnd = CreateWindow("HelloClass", "HelloSDK Application",
   WS_OVERLAPPEDWINDOW,
   CW_USEDEFAULT, CW_USEDEFAULT,
   CW_USEDEFAULT, CW_USEDEFAULT,
   NULL, NULL, hInstance, NULL);
ShowWindow(hwnd, nCmdShow); // 윈도우가 화면에 나타나게함
UpdateWindow(hwnd); // 윈도우 프로시저가 WM_PAINT 메시지를 처리하게 함
```


HelloSDK 예제 분석

- 메시지 루프
  - 메시지 대기열 (Message Queue)에서 메시지를 하나씩 꺼내서 처리
  - GetMessage - 메시지 관리를 위한 구조체
    - Message Queue에서 메시지 하나를 꺼내서 메시지 변수에 저장
    - Message가 WM_QUIT이면 0을 return하고 while loop 종료
    - GetMessage("message 관리 구조체", "메시지 핸들", "최소", "최대")
  - TranslateMessage – Keyboard 관련 메시지를 처리
  - DispatchMessage – 해당 메시지를 Window procedure에 보냄

```
while(GetMessage(&msg, NULL, 0, 0) > 0){
   TranslateMessage(&msg);
   DispatchMessage(&msg);
}
return msg.wParam;
```


HelloSDK 예제 분석

- 윈도우 프로시저 WndProc
  - 윈도우 메시지를 처리하는 핵심 함수

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

  - hwnd – 윈도우 핸들. 어느 윈도우가 window procedure를 호출했는지 구분
  - message – 구체적인 메시지 종류, WM_ 으로 시작
  - wParam, lParam – 메시지 종류에 따라 부가적인 정보전달
  - CALLBACK – OS나 컴파일러 환경에 따라서 파라미터나 리턴값을 넘겨주고 넘겨받는 방식을 정하는 MACRO
  - LRESULT – long type과 동일

HelloSDK 예제 분석

- 윈도우 프로시저 WndProc

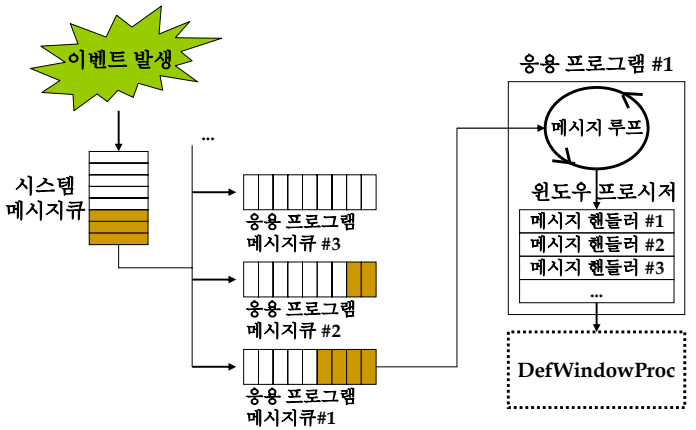
```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message,
   WPARAM wParam, LPARAM lParam)
{
   HDC hdc;
   PAINTSTRUCT ps;
   char *str = "Hello, SDK";

   // 메시지 종류에 따라 응용 프로그램이 적절히 처리한다.
   switch(message) {
   case WM_CREATE:
      return 0;
   case WM_PAINT:
      hdc = BeginPaint(hwnd, &ps);
```

HelloSDK 예제 분석

```
        TextOut(hdc, 100, 100, str, lstrlen(str));
        EndPaint(hwnd, &ps);        // Device Context 해제
        return 0;
    case WM_LBUTTONDOWN:
        MessageBox(hwnd, "마우스를 클릭했습니다.",
                         "마우스 메시지", MB_OK);
        return 0;
    case WM_DESTROY:
        PostQuitMessage(0);         // 메시지큐에 WM_QUIT가 들어간다
        return 0;
    }

    // 응용 프로그램이 처리하지 않으면 윈도우 운영체제가 처리한다.
    return DefWindowProc(hwnd, message, wParam, lParam);
}
```

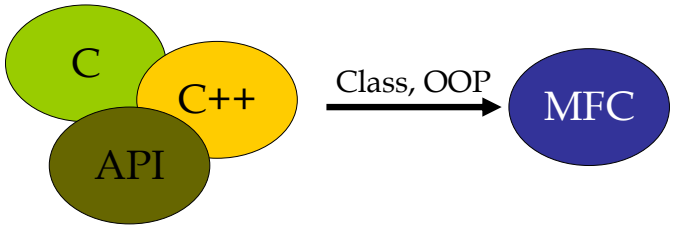

HelloSDK 예제 분석

- 요약



WIN32 API와 MFC

- MFC
  - Win32 API를 이용해서 프로그램을 작성할 경우 코드의 길이가 매우 길어지게 되며, 어떠한 부분에 있어서는 모든 프로그래머들이 반복적으로 작성해야 하는 부분도 있음.
  - 이러한 단점을 보완하기 위해서 Microsoft의 수많은 개발자들이 클래스 기반의 견고한 코드를 클래스화해서 만든 라이브러리임.



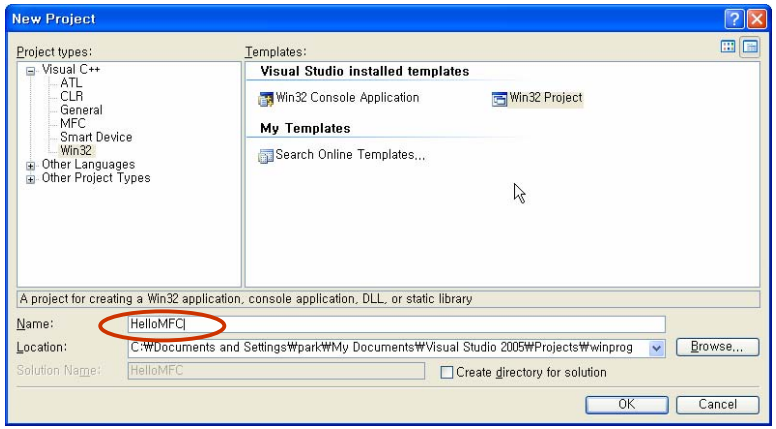
MFC (Microsoft Foundation Class)

- SDK (Windows API)의 WinMain() 함수의 기능과 Window Procedure 기능을 적절히 분할하여 2개의 클래스 제공

응용 프로그램 자신만의 특징을 갖기 위하여 위의 2 클래스를 그대로 사용하지 않고, OOP의 inheritance기능을 이용

HelloMFC 예제 작성

- 프로젝트 생성

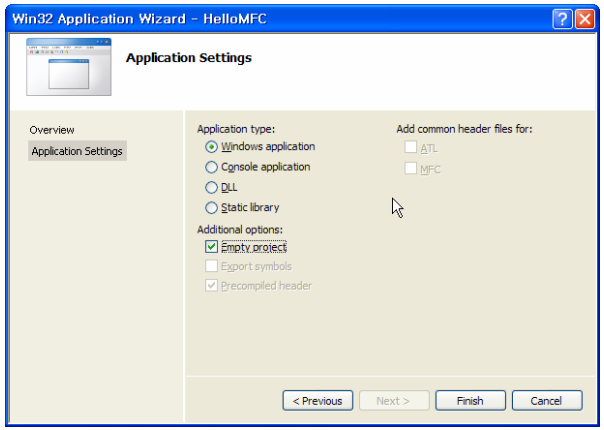


HelloMFC 예제 작성

- 1단계 옵션 설정

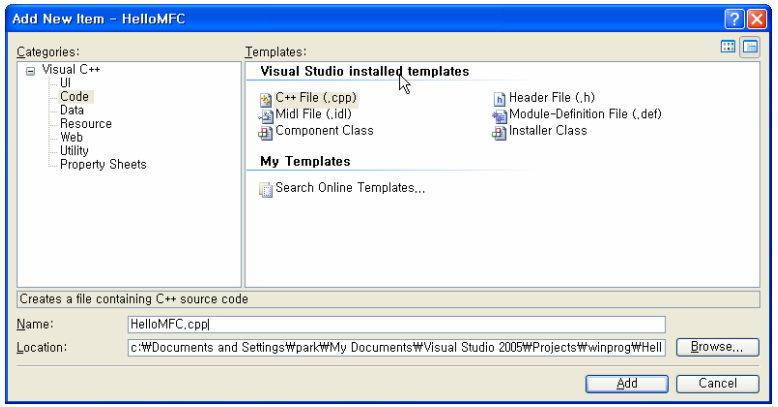
HelloMFC 예제 작성

- 1단계 옵션 설정



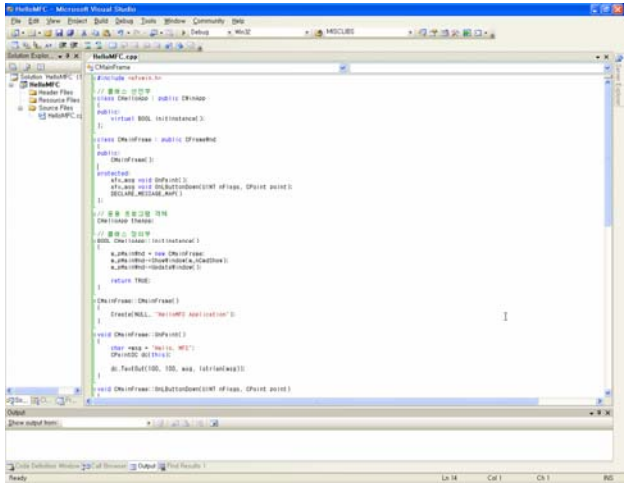
HelloMFC 예제 작성

- 소스 파일 추가



HelloMFC 예제 작성

- 코드 입력



HelloMFC 예제 작성

- 코드 입력

```
#include <afxwin.h>

// 클래스 선언부
class CHelloApp : public CWinApp
{
public:
   virtual BOOL InitInstance();
};

class CMainFrame : public CFrameWnd
{
public:
   CMainFrame();
```


HelloMFC 예제 작성

```
protected:
   afx_msg void OnPaint();
   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
   DECLARE_MESSAGE_MAP()
};

// 응용 프로그램 객체
CHelloApp theApp;

// 클래스 정의부
BOOL CHelloApp::InitInstance()
{
   m_pMainWnd = new CMainFrame;
   m_pMainWnd->ShowWindow(m_nCmdShow);
```


HelloMFC 예제 작성

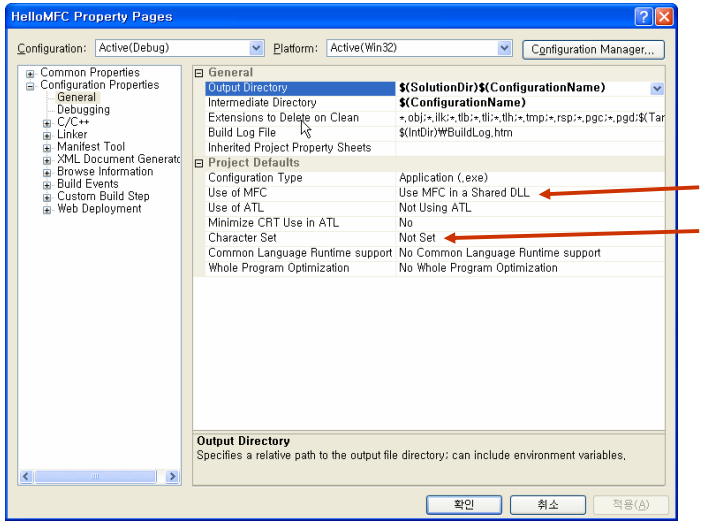
```
   m_pMainWnd->UpdateWindow();
   return TRUE;
}

CMainFrame::CMainFrame()
{
   Create(NULL, "HelloMFC Application");
}

void CMainFrame::OnPaint()
{
   char *msg = "Hello, MFC";
   CPaintDC dc(this);
```

HelloMFC 예제 작성

```
    dc.TextOut(100, 100, msg, lstrlen(msg));
}

void CMainFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
    MessageBox("마우스를 클릭했습니다.", "마우스 메시지");
}

// 메시지맵
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_PAINT()
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
```


HelloMFC 예제 작성

- 프로젝트 설정 변경



HelloMFC 예제 작성

- F7 (Build Solution) -> F5 (Start Debugging)
- 실행 화면

HelloMFC 예제 분석

- 헤더 파일

```
#include <afxwin.h>
```

SDK에서 #include <windows.h>를
MFC에서는 #include <afxwin.h>로

- 클래스 선언부

```
class CHelloApp : public CWinApp
{
public:
    virtual BOOL InitInstance();
};
```

HelloMFC 예제 분석

- 클래스 선언부

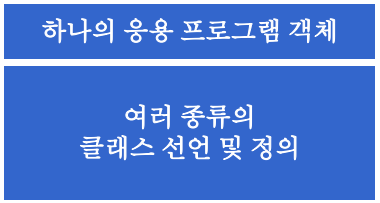
```
class CMainFrame : public CFrameWnd
{
public:
    CMainFrame();

protected:
    afx_msg void OnPaint();
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    DECLARE_MESSAGE_MAP()
};
```


HelloMFC 예제 분석

- 응용 프로그램 객체
  - 모든 MFC 프로그램은 CWinApp 클래스에서 하나의 클래스를 파생하고 이 클래스로부터 하나의 객체를 전역 변수로 생성
  - WinMain 함수는 MFC에서 제공

```
CHelloApp theApp;
```

  - MFC 응용 프로그램의 구조



HelloMFC 예제 분석

- 클래스 정의부
  - 숨겨진 WinMain()은 프로그램이 시작되면 처음으로 InitInstance함수를 호출 – 초기화에 사용
  - Virtual 함수로 선언해야 함 – 재정의하지 않으면 CWinApp에 있는 InitInstance가 수행됨
  - ShowWindow()에는 윈도우 핸들이 존재하지 않음 – 윈도우 핸들을 내부에 숨기고 있기 때문에

```
BOOL CHelloApp::InitInstance()
{
    m_pMainWnd = new CMainFrame;
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}
```

```
Class CHelloApp: public CWinApp {
    public:
        virtual BOOL InitInstance();
};
```

HelloMFC 예제 분석

- 클래스 정의부
  - SDK의 CreateWindow() 함수에 상응하는 역할

```
CMainFrame::CMainFrame()
{
    Create(NULL, "HelloMFC Application"); // 윈도우 생성
}
```

HelloMFC 예제 분석

- 클래스 정의부

```
void CMainFrame::OnPaint()      // SDK에서 WM_PAINT 역할
{
   char *msg = "Hello, MFC";
   CPaintDC dc(this);

   dc.TextOut(100, 100, msg, lstrlen(msg));
}

// SDK에서 WM_LBUTTONDOWN 역할
void CMainFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
   MessageBox("마우스를 클릭했습니다.", "마우스 메시지");
}
```


HelloMFC 예제 분석

- 메시지맵
  - 윈도우 메시지와 해당 메시지 핸들러를 연결시키는 MFC 방법
  - ON_WM_PAINT() – WM_PAINT와 OnPaint()를 연결
  - ON_WM_LBUTTONDOWN() – WM_LBUTTONDOWN과 OnLButtonDown()를 연결

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_PAINT()
   ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
```


HelloMFC 예제 분석

- 숨겨진 MFC의 WinMain

```
WinMain( ) // MFC의 내부에 숨겨진 프로그램 실행의 시작점
{
   // ptr은 포인터로서 응용 프로그램 객체의 주소 값을 가지고 있다.
   ...
   ptr->InitInstance( );  // 초기화: 프레임 윈도우 객체를 생성한다.
                          // 프레임 윈도우 객체의 생성자에서
                          // 실제 윈도우가 만들어진다.
   ptr->Run( );  // 메시지 루프: 프레임 윈도우에게 메시지를 보낸다.
                 // 프레임 윈도우가 받은 메시지의 종류에 따라
                 // 해당 메시지 핸들러가 적절히 호출된다.
   ptr->ExitInstance( ); // 종료: 각종 청소 작업을 수행한다.
   ...
}
```


Coding Rules

- 변수 표기법
  - 의미있는 단어의 사용
  - 멤버변수는 "m_"라는 접두어 추가
  - 헝가리안 표기법

m_lpszFilename

m_ : 멤버 변수
lp : 포인터 변수
sz : 널로 끝나는 문자열
Filename: 파일이름

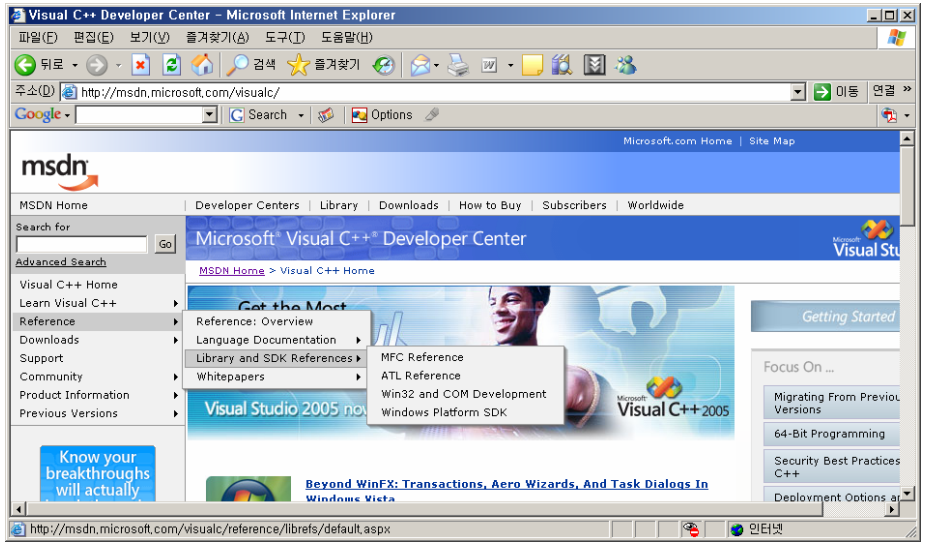
접두어	의미
b	BOOL형 변수
d	double형 변수
h	HANDLE형 변수
n	int형 변수
p 또는 lp	pointer 변수
sz	NULL 문자로 끝나는 문자열
u	unsigned int형 변수
w	WORD(unsigned short형 변수)
dw	DWORD(unsigned long형 변수)
str	CString형 변수
clr	COLORREF

Windows Programming

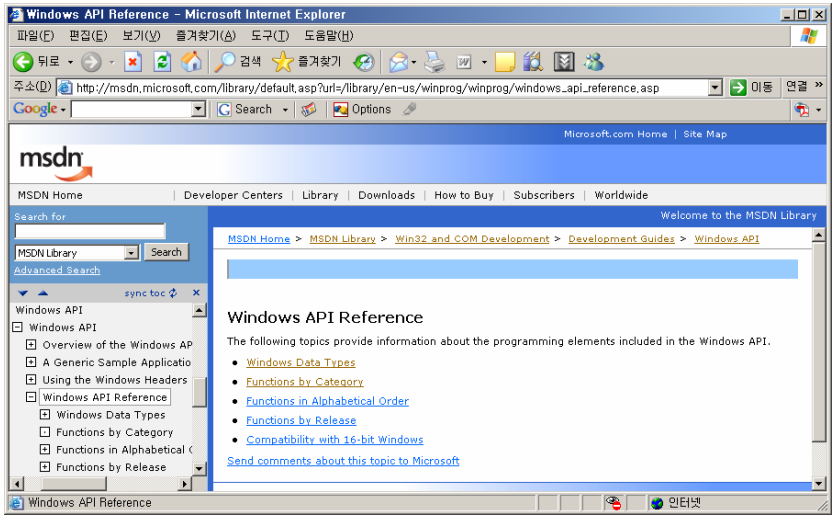
- 대표적인 윈도우 메시지

윈도우 메시지	발생상황	메시지 처리기
WM_CREATE	윈도우가 생성될 때	OnCreate
WM_ACTIVATE	윈도우가 활성화되거나 비활성화될 때	OnActivate
WM_PAINT	윈도우가 다시 그려져야 할 때	OnPaint
WM_MOUSEMOVE	마우스커서가 움직였을 때	OnMouseMove
WM_COMMAND	사용자가 메뉴 등을 명령으로 내렸을 때	OnDestroy
WM_LBUTTONDOWN	마우스 왼쪽버튼을 눌렀을 때	OnLButtonDown
WM_LBUTTONUP	마우스 왼쪽 버튼을 떼었을 때	OnLButtonUp
WM_LBUTTONDBLCILK	마우스 왼쪽 버튼이 더블 클릭됐을 때	OnLButtonDblClk
WM_KEYDOWN	키보드가 눌렸을 때	OnKeyDown
WM_KEYUP	키보드가 떼었을 때	OnKeyUp
WM_SIZE	윈도우의 크기가 변경되었을 때	OnSize
WM_MOVE	윈도우가 이동되었을 때	OnMove
WM_TIMER	설정된 타이머 시간이 다 되었을 때	OnTimer
WM_DESTROY	윈도우가 없어질때	OnDestroy

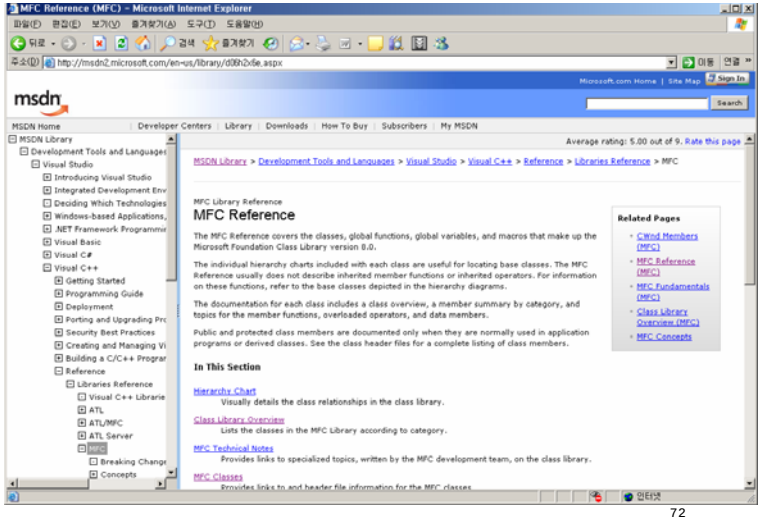
MSDN Reference



Windows API Reference



MFC Reference



Practice

- HelloSDK와 HelloMFC 프로그램을 이용하여 이름, 학과, 학번, 및 자기소개를 출력하는 프로그램을 작성하라
  - WM_PAINT 와 OnPaint를 비교한다
  - WM_LBUTTONDOWN 과 OnLButtonDown를 비교한다