DSC공유대학
Daejeon Sejong Chungnam Regional Innovation Platform

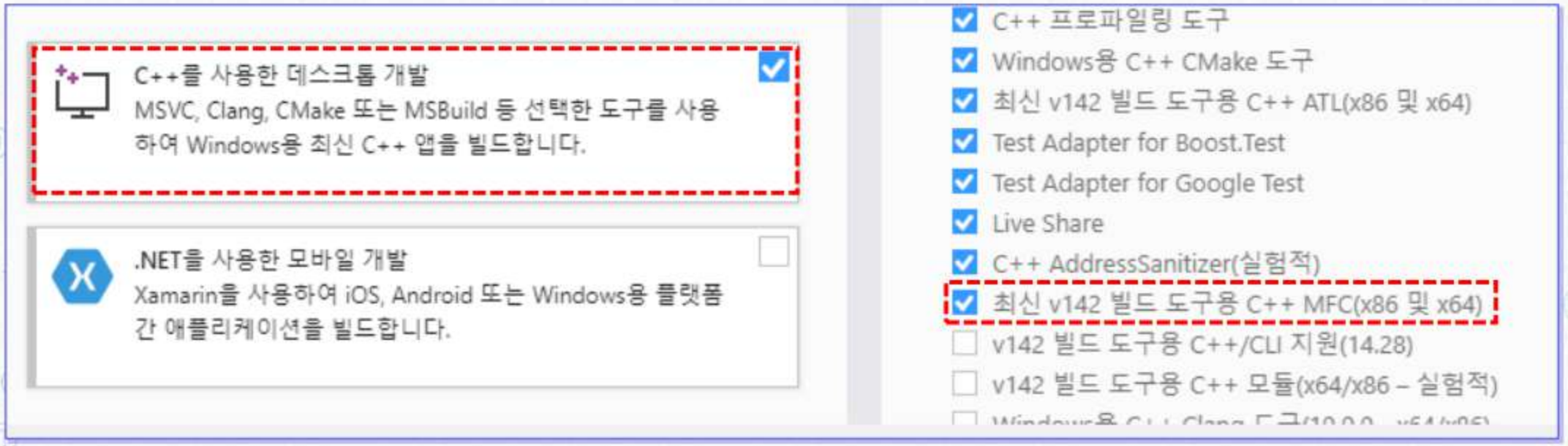
모빌리티 윈도우 프로그래밍

01. 윈도우 프로그래밍의 이해

선문대학교 지능형전장제어시스템 김창성 교수

Visual Studio 2019 추가 옵션 설치

Visual Studio Installer



학습 내용

윈도우 프로그래밍의 기본 개념

Win32 SDK를 이용하여 윈도우 프로그램을 작성하는 방법

윈도우 프로그램의 기본 형식

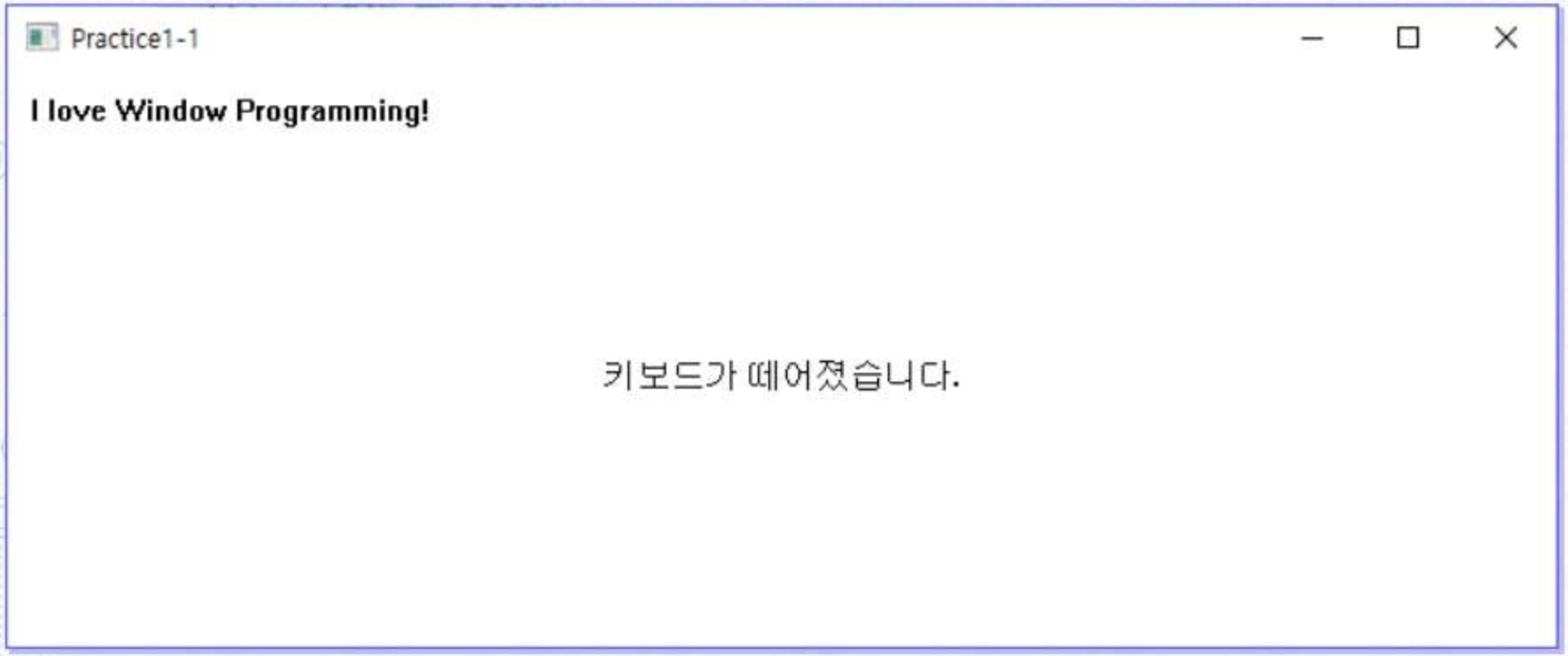
목차

실습 내용 미리 보기

Win32 SDK를 사용하여 윈도우 만들기

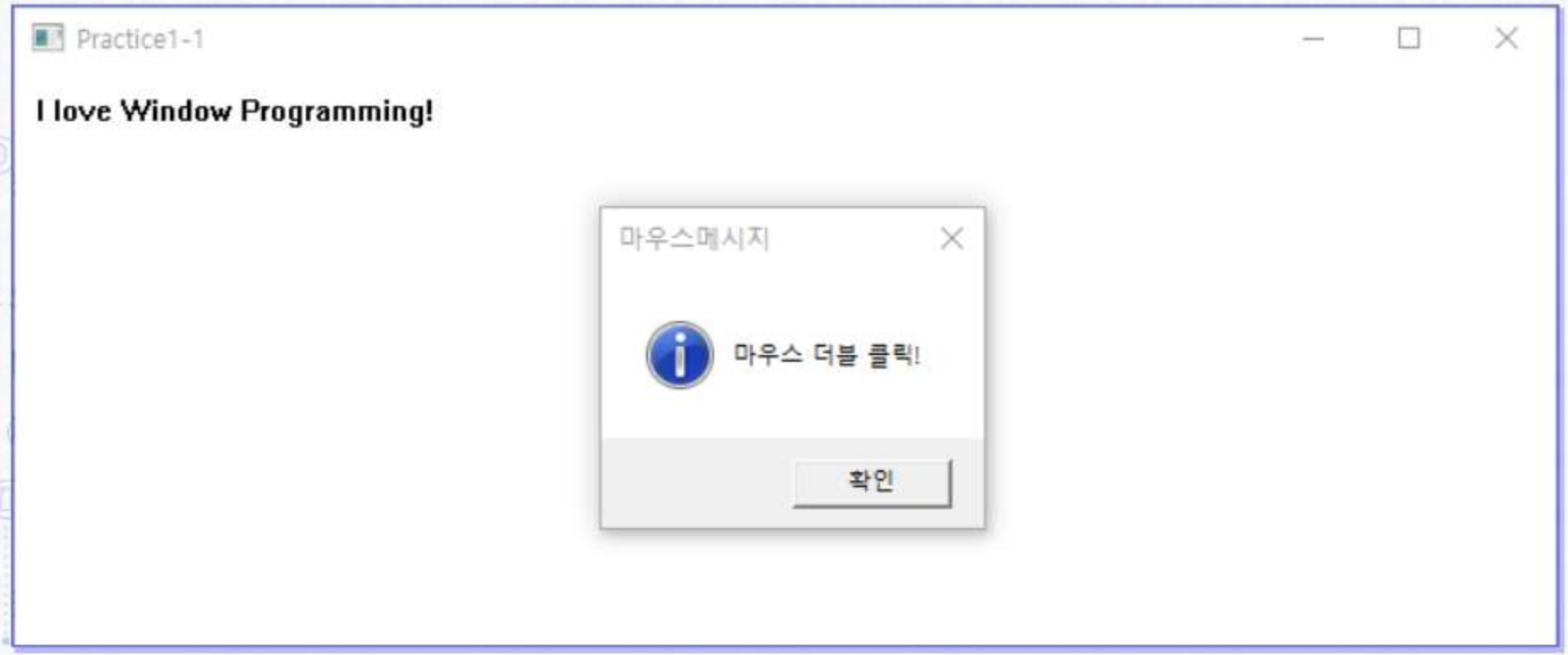
실습 내용 미리 보기

키보드의 키를 뗀 경우



실습 내용 미리 보기

마우스 좌 더블 클릭



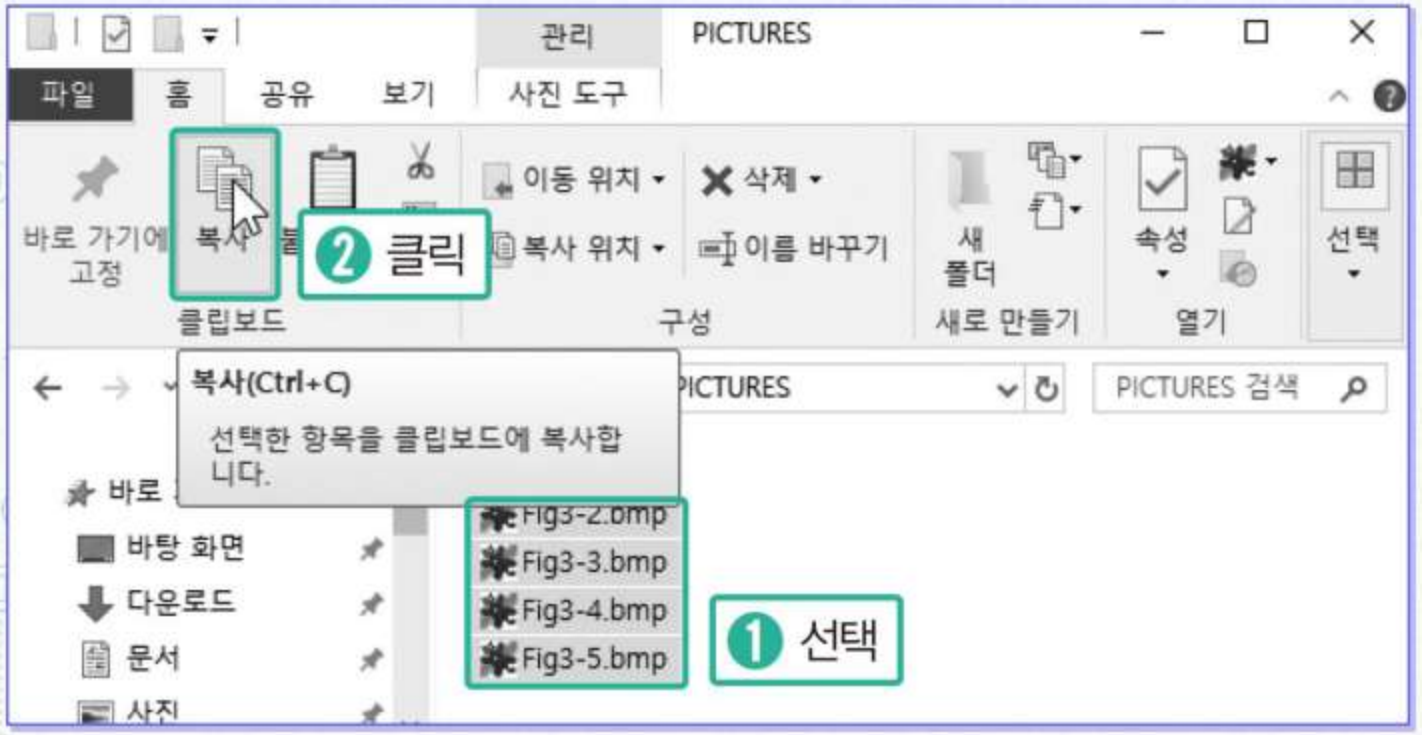
1. 윈도우 운영체제의 특징

텍스트 기반 운영체제(DOS)

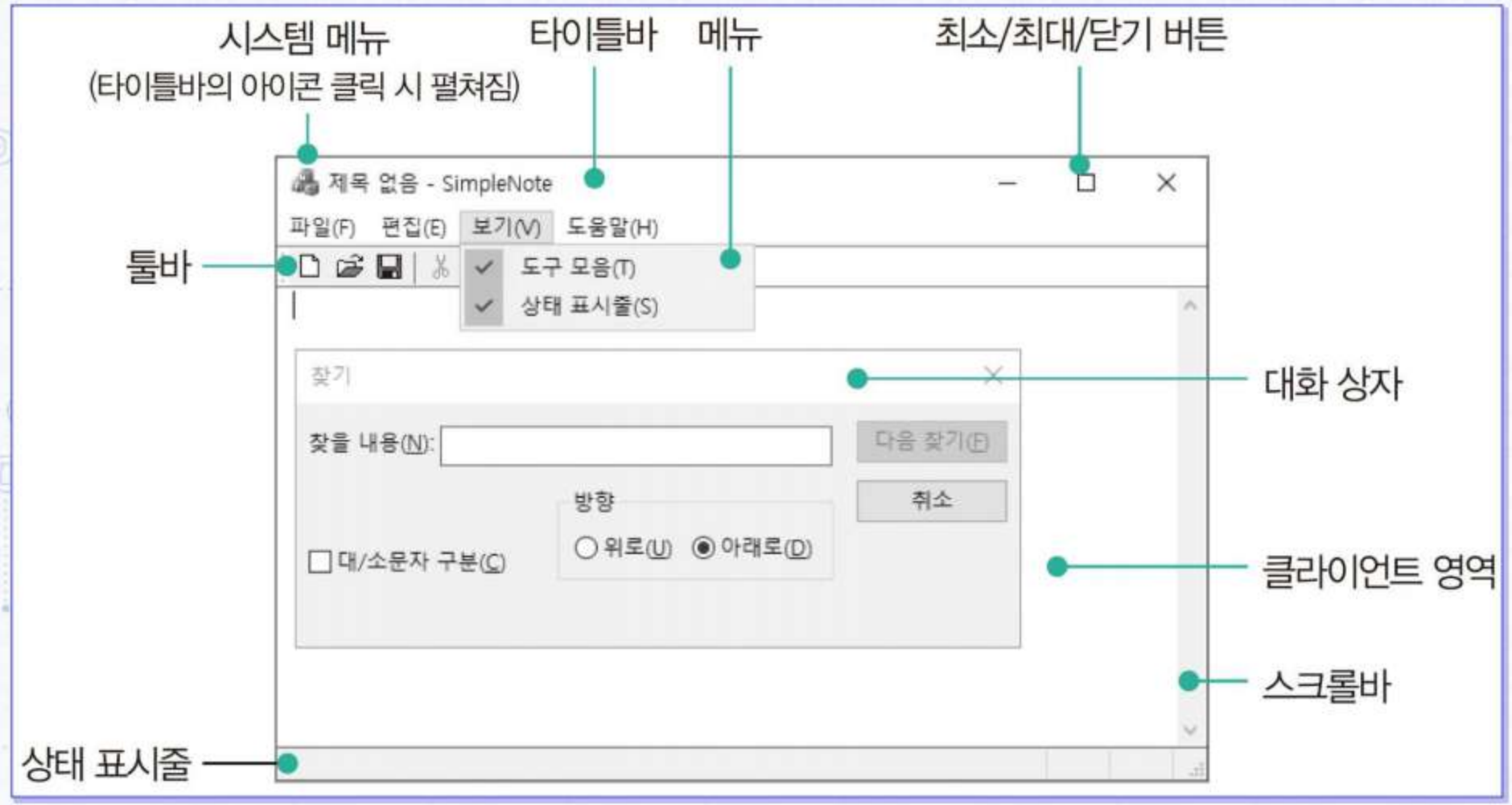
```
C:\PICTURES>COPY *.BMP C:\TEMP
Fig3-1.bmp
Fig3-2.bmp
Fig3-3.bmp
Fig3-4.bmp
Fig3-5.bmp
        5 file(s) copied.

C:\PICTURES>_
```

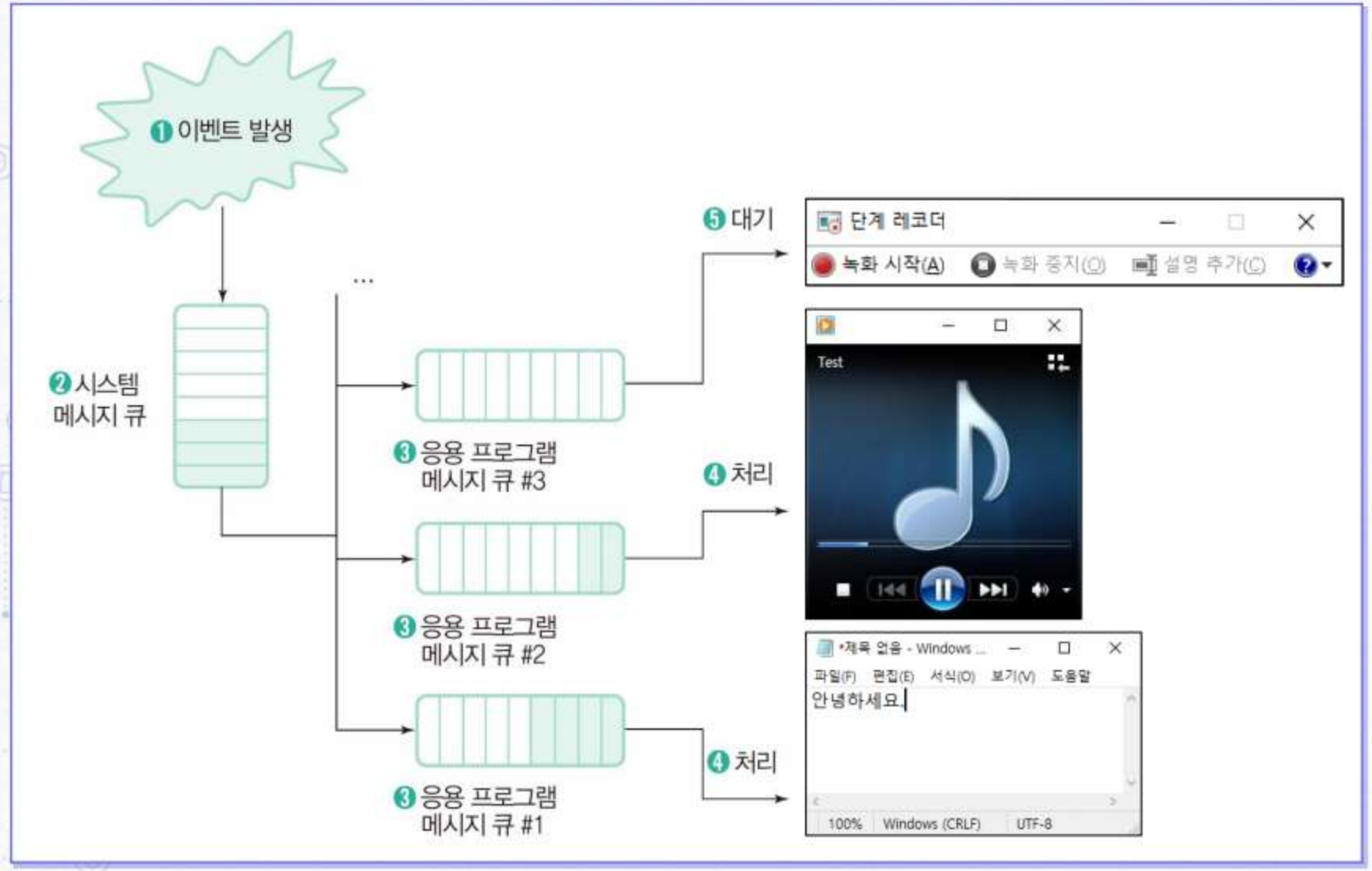
그래픽 사용자 인터페이스(GUI)



사용자 인터페이스의 구성 요소

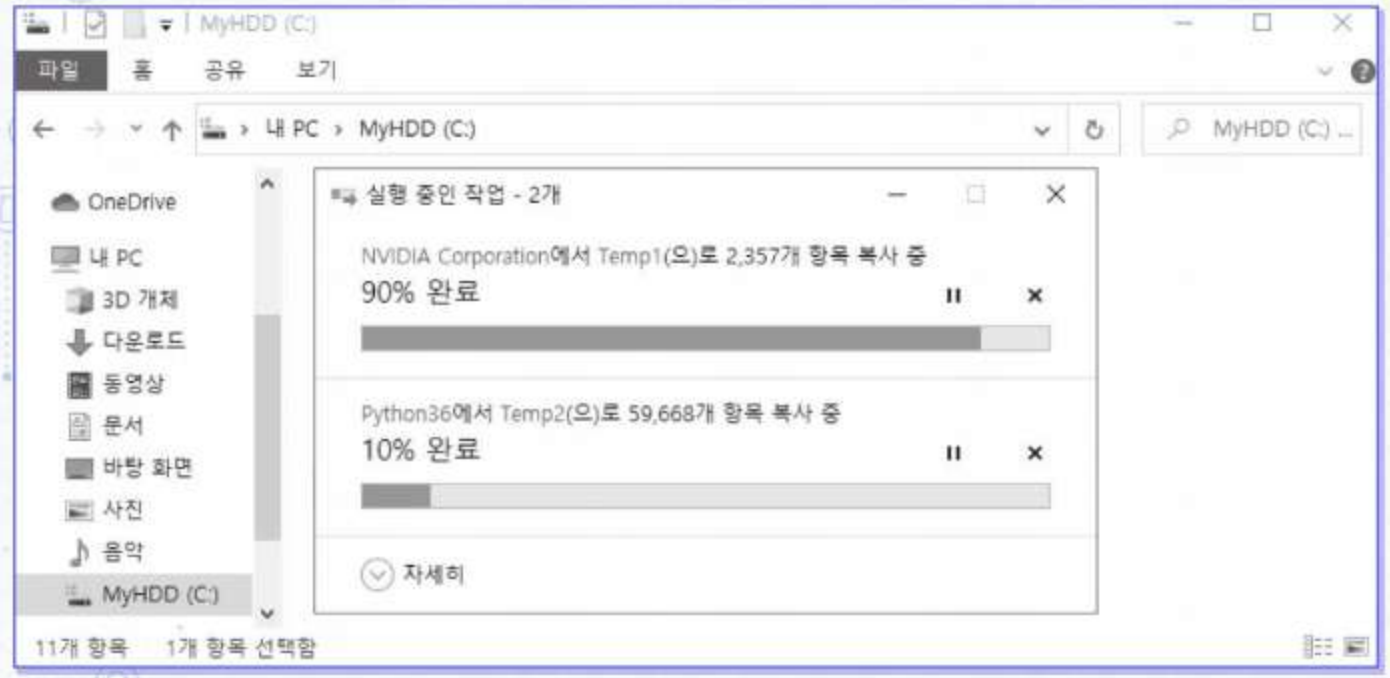


메시지 구동 구조


1 이벤트 발생
2 시스템
메시지 큐
...
3 응용 프로그램
메시지 큐 #3
3 응용 프로그램
메시지 큐 #2
3 응용 프로그램
메시지 큐 #1
5 대기
4 처리
4 처리
단계 레코더
녹화 시작(A)
녹화 중지(O)
설명 추가(C)
Test
*제목 없음 - Windows ...
파일(F) 편집(E) 서식(O) 보기(V) 도움말
안녕하세요.
100%
Windows (CRLF)
UTF-8

멀티태스킹과 멀티스레딩

멀티태스킹(Multitasking)	운영체제가 여러 개의 응용 프로그램을 동시에 실행
멀티스레딩(Multithreading)	응용 프로그램 내부에서 여러 개의 실행 흐름(=스레드)을 동시에 진행



윈도우 응용 프로그램의 특징

API 호출문 집합

- 윈도우 운영체제가 응용 프로그램을 위해 제공하는 각종 함수의 집합
  - 16비트 윈도우 → Win16 API
  - 32 또는 64비트 윈도우 → Win32 API

MFC

- 내부적으로 API를 호출하여 많은 기능을 구현

윈도우 응용 프로그램의 많은 부분은 API를 호출하여 구현

윈도우 응용 프로그램의 특징

메시지 핸들러 집합

메시지 핸들러(Message Handler)

- 메시지를 받았을 때 동작을 결정하는 코드
  - 키보드 메시지 핸들러
  - 마우스 메시지 핸들러
  - 메뉴 메시지 핸들러

윈도우 프로시저(Window Procedure)

- 메시지 핸들러의 집합

실행 파일과 DLL 집합

- DLL(Dynamic-Link Library)
  - 프로그램 실행 중에 결합하여 사용할 수 있는 코드와 리소스
  - 파일 확장자 : ".DLL"
- 윈도우 운영체제가 제공하는 API는 DLL 형태로 제공
- 프로그래머가 직접 필요한 기능을 DLL로 제작

장치 독립성(Device-Independency)

- 주변 장치가 바뀔 경우
  - 장치 드라이버(Device Driver)만 설치하면 프로그램을 수정하지 않고 실행할 수 있음
- 응용 프로그램은 API를 통해 장치 드라이버와 간접적으로 통신 → 장치 독립성을 가짐

2. 윈도우 응용프로그램 개발 방식

프로그래밍 언어 학습의 개발 환경

C, C++ 프로그래밍 언어에 대한 학습

- 언어에 대한 이해가 필요
- 개발 환경
  - UNIX
    - gcc
  - MicroSoft Windows
    - 간단한 IDE (개발환경, Integrated Development Environment)를 사용

윈도우 응용 프로그램의 개발 방식

SDK(Software Development Kit)

- MS에서 윈도우 응용 프로그램 제작을 위해 배포하는 개발 도구 모음
- 컴파일러를 비롯한 각종 개발툴, 헤더 파일, 라이브러리 파일, 도움말 등을 포함
- SDK 프로그램 개발
  - C/C++ 언어로 윈도우 API를 직접 호출해서 프로그램을 구현

장점	• API를 직접 다루기 때문에 세부 제어가 가능함 • 윈도우 운영체제가 제공하는 모든 기능을 사용 가능 • 생성 코드의 크기가 작고 속도도 빠름

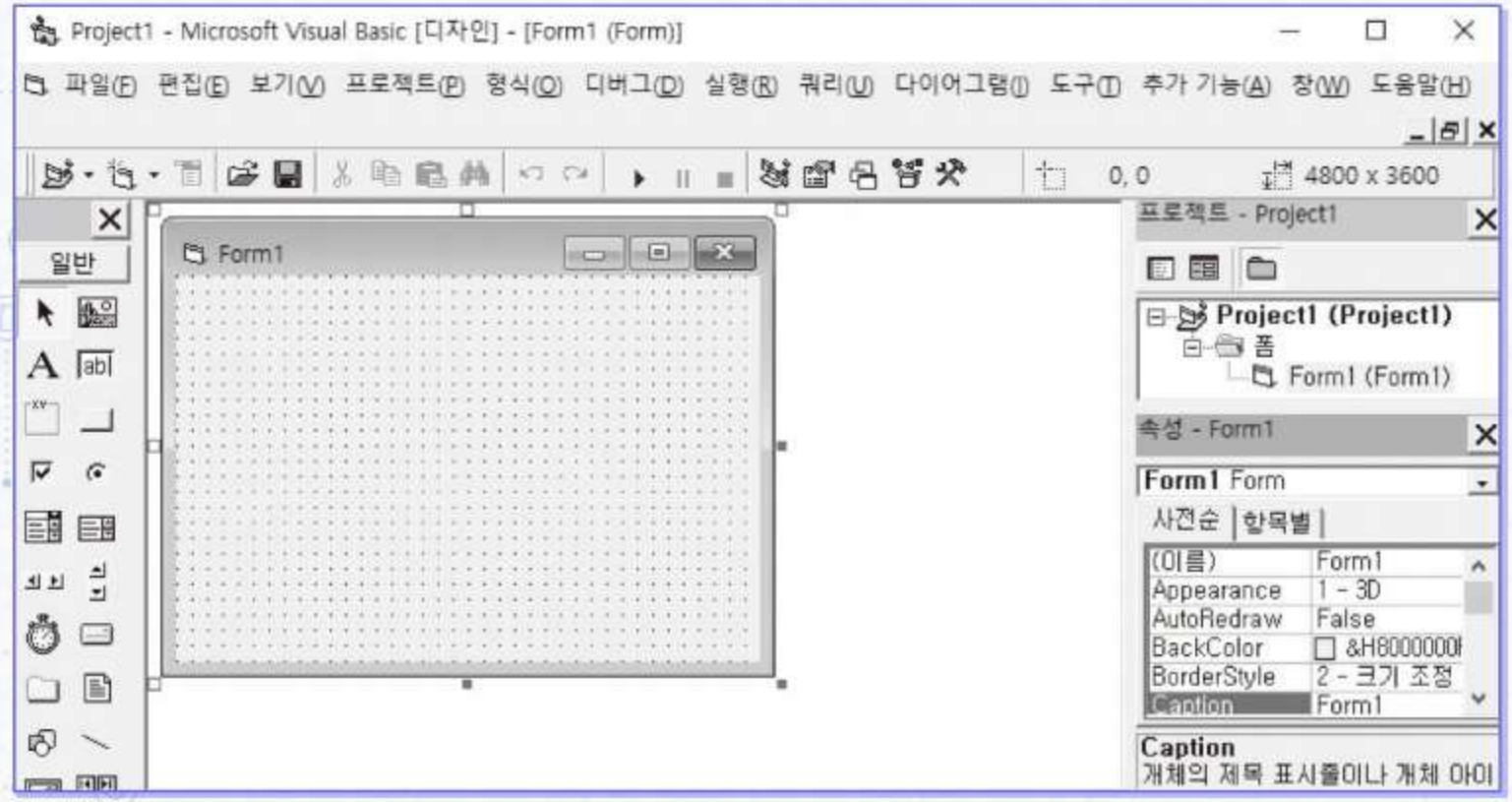
단점	• 다른 개발 방식에 비해 생산성이 매우 낮음

윈도우 응용 프로그램의 개발 방식

RAD(Rapid Application Development)

화면을 시각적으로 디자인하고 코드를 추가해 프로그램을 개발

- 비주얼 베이직(Visual Basic)
- 델파이(Delphi)



RAD(Rapid Application Development)

장점	• 직관적으로 간편하게 프로그래밍할 수 있다. • 빠른 시간 내에 원하는 기능의 프로그램 개발 가능하다.
단점	• SDK나 클래스 라이브러리를 이용한 개발 방식보다 생성 코드의 크기가 크고 실행 속도도 떨어지는 편이다. • 윈도우 운영체제가 제공하는 모든 기능을 활용해 세부적으로 제어가 필요한 경우 개발하기 어려울 수도 있다.

클래스 라이브러리(Class Library)

- 윈도우 응용 프로그램 개발에 필수적인 기능을 C++와 같은 객체지향 언어를 이용하여 클래스화한 것(MFC, OWL)

장점	• SDK를 이용한 방식보다 빠른 속도로 개발할 수 있다. • API를 직접 사용해서 세부적으로 제어할 수 있다. • RAD 개발 방식보다 코드 크기와 실행 속도 면에서 유리하다.
단점	• 객체지향 프로그래밍에 익숙해야 한다. • 클래스 라이브러리의 구조와 각 클래스의 기능 및 관계를 파악하기 위한 초기 학습 기간이 긴 편이다.

.Net 프레임워크

- 윈도우 운영체제에 설치할 수 있는 소프트웨어 개발 및 실행 환경
- 특징
  - 공용 언어 런타임(CLR, Common Language Runtime)이라는 소프트웨어 가상 머신을 제공한다.
  - 가상 머신 제어 하에서 응용 프로그램이 구동된다.(장치 독립성)
  - 윈도우 API에 버금가는 방대한 라이브러리를 제공한다.
  - 언어에 상관없이 라이브러리를 사용할 수 있다.(언어 독립성)

윈도우 응용 프로그램의 개발 방식

SDK

C언어로 윈도우 API를
사용할 수 있게 제공하는 라이브러리

MFC

C++ 클래스 형태로 캡슐화해서
만든 클래스 라이브러리

효율적인 윈도우 응용프로그램 개발 방법

- 기본 틀 : MFC를 사용하여 간단히 처리
- 중요하게 처리할 부분 : SDK를 활용하여 작성

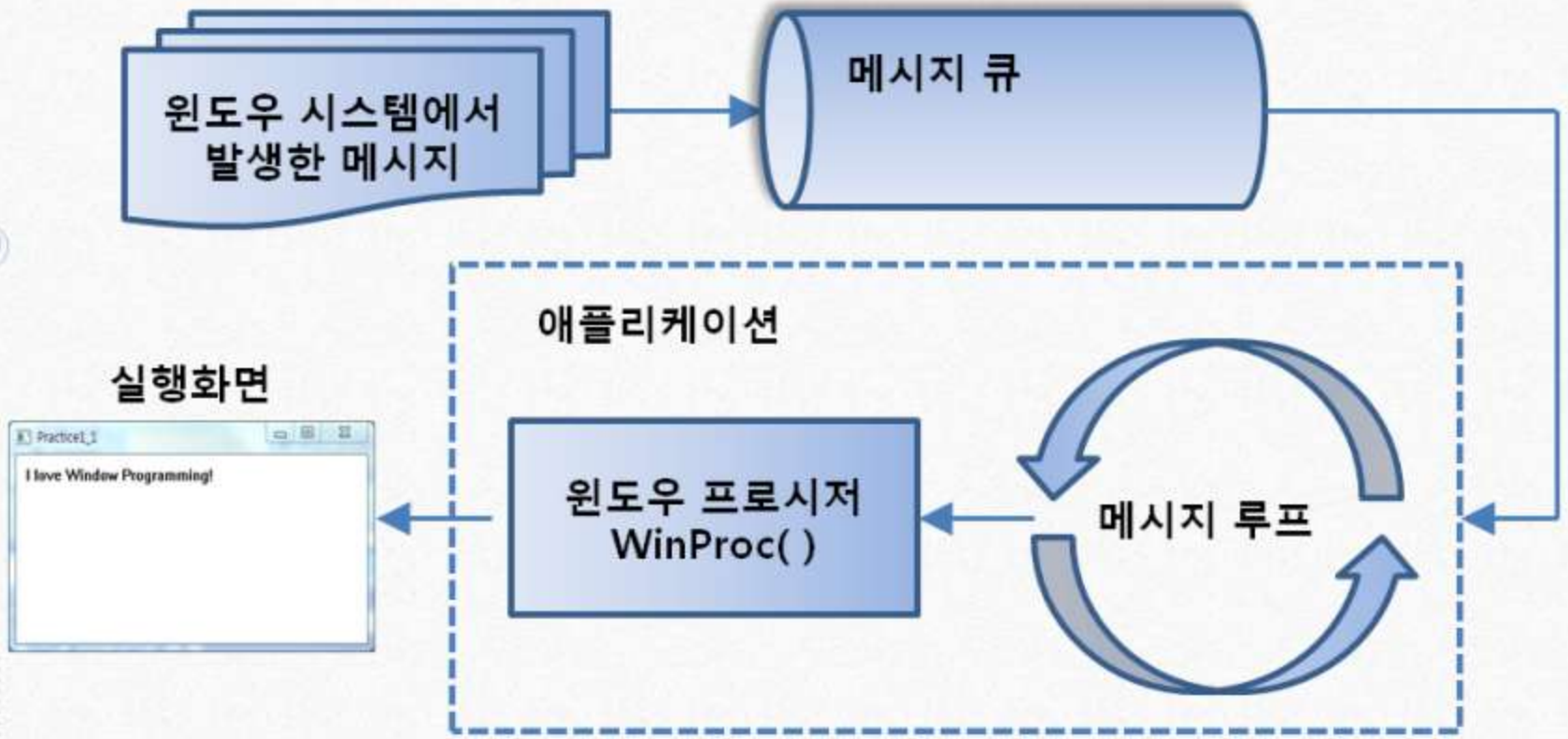
3. 윈도우 프로그래밍의 개념

사용자가 발생시키는
이벤트에 대하여 처리를 수행

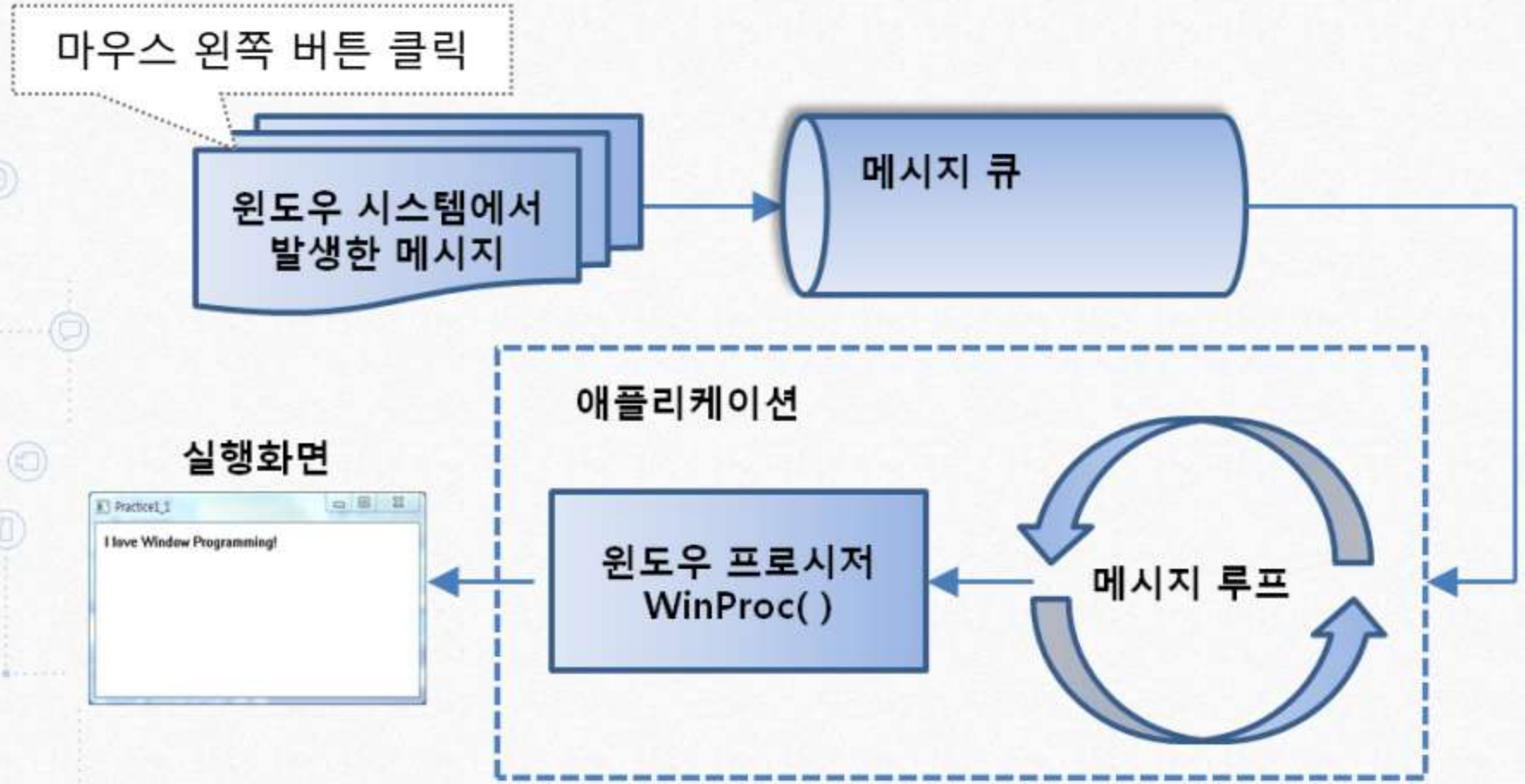
프로그래밍 방법

메시지 기반(Message Driven)
이벤트 기반(Event Driven)

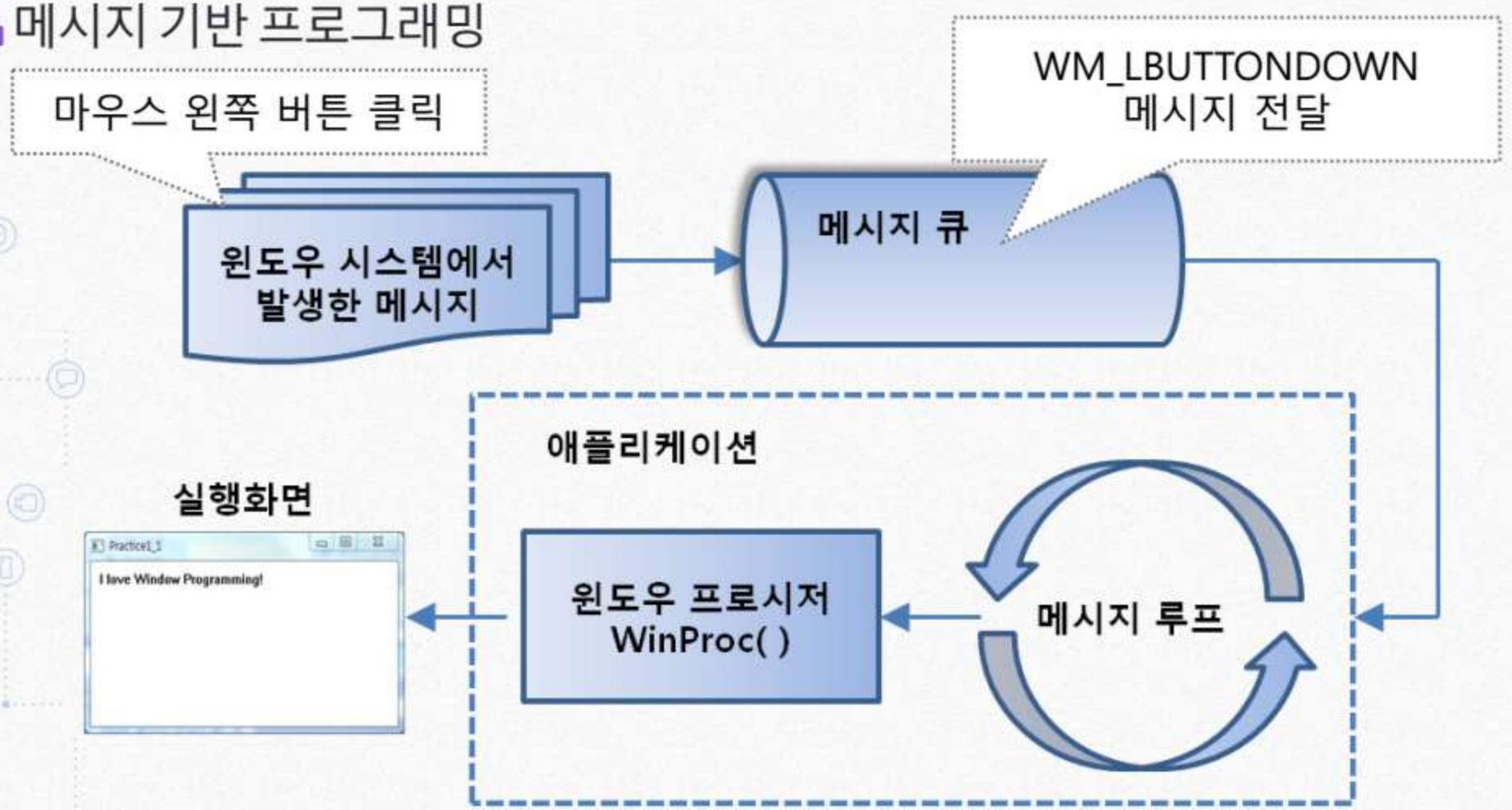
메시지 기반 프로그래밍


윈도우 시스템에서
발생한 메시지
메시지 큐
애플리케이션
실행화면
Practice1_1
I love Window Programming!
윈도우 프로시저
WinProc( )
메시지 루프

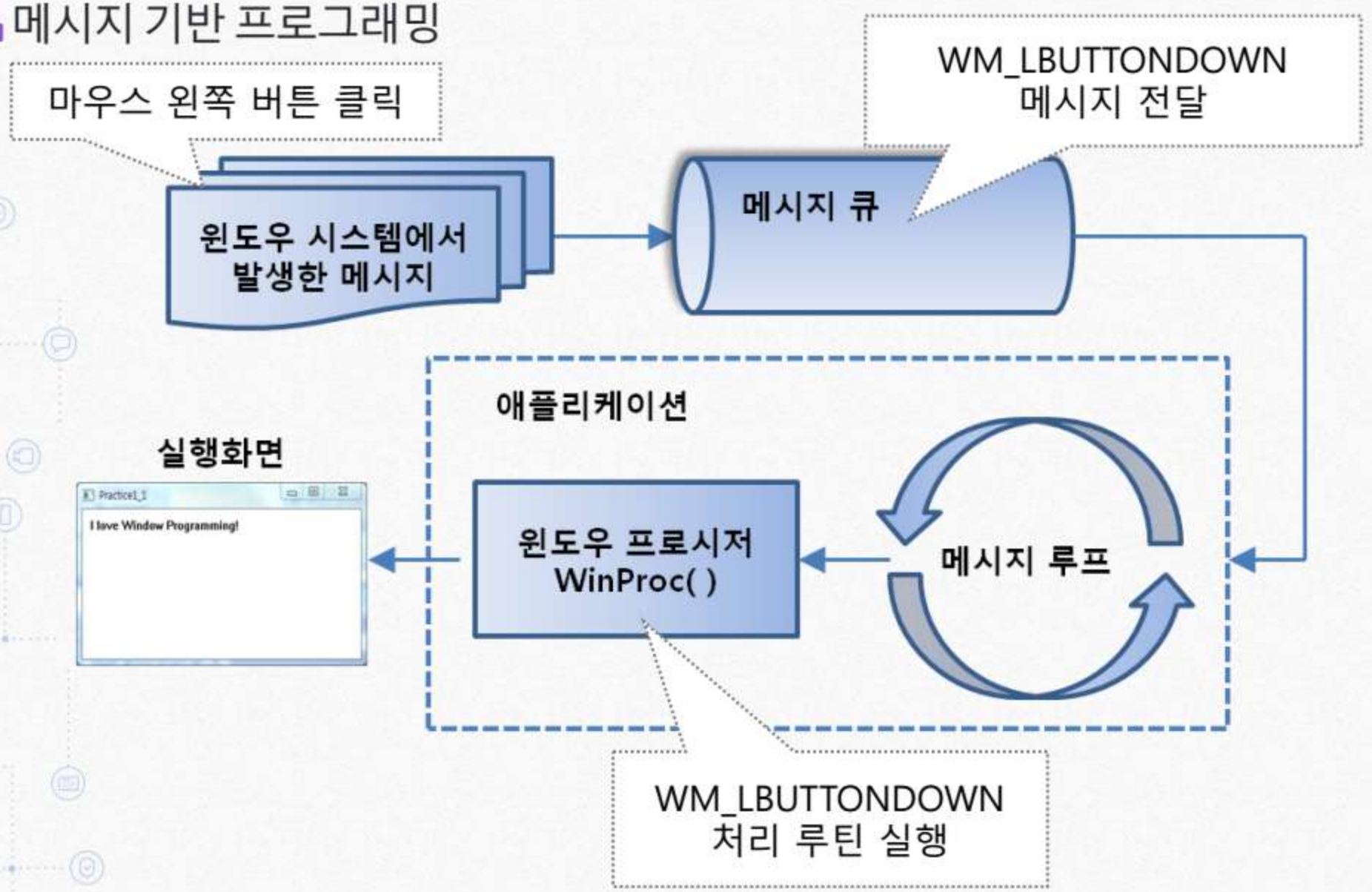
메시지 기반 프로그래밍


마우스 왼쪽 버튼 클릭
윈도우 시스템에서
발생한 메시지
메시지 큐
애플리케이션
실행화면
Practice1_1
I love Window Programming!
윈도우 프로시저
WinProc( )
메시지 루프

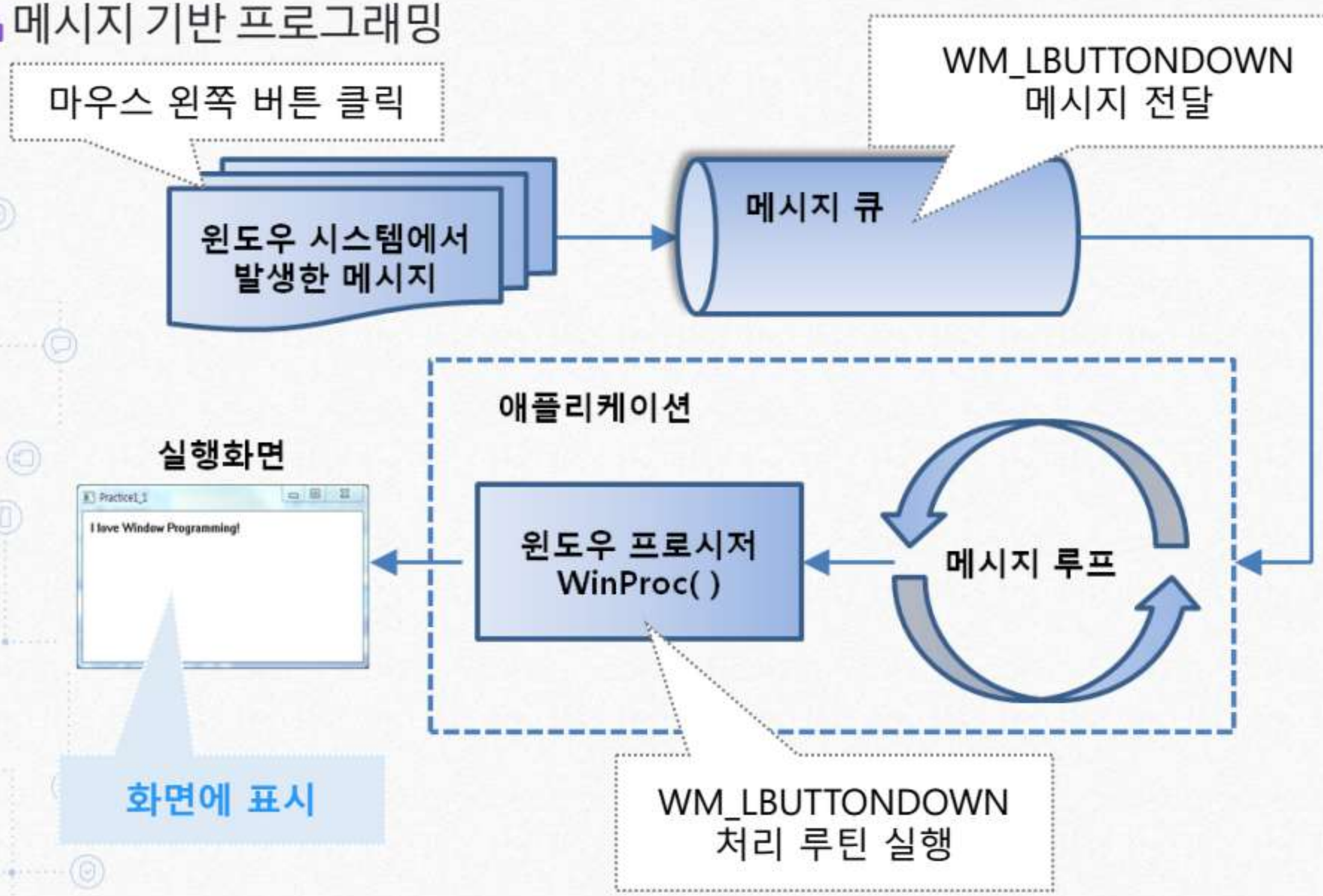
메시지 기반 프로그래밍

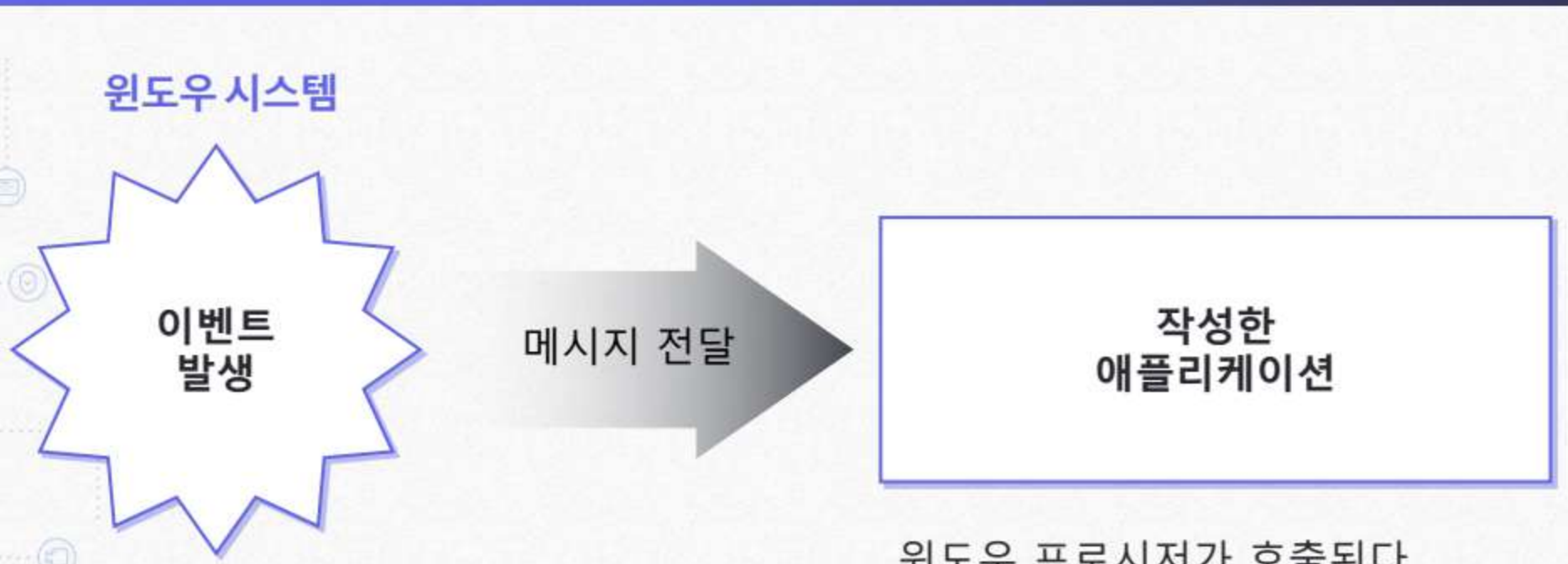

마우스 왼쪽 버튼 클릭
WM_LBUTTONDOWN
메시지 전달
윈도우 시스템에서
발생한 메시지
메시지 큐
애플리케이션
실행화면
Practice1_1
I love Window Programming!
윈도우 프로시저
WinProc( )
메시지 루프

메시지 기반 프로그래밍


마우스 왼쪽 버튼 클릭
WM_LBUTTONDOWN
메시지 전달
윈도우 시스템에서
발생한 메시지
메시지 큐
애플리케이션
실행화면
Practice1_1
I love Window Programming!
윈도우 프로시저
WinProc( )
메시지 루프
WM_LBUTTONDOWN
처리 루틴 실행

메시지 기반 프로그래밍


마우스 왼쪽 버튼 클릭
WM_LBUTTONDOWN
메시지 전달
윈도우 시스템에서
발생한 메시지
메시지 큐
애플리케이션
실행화면
Practice1_1
I love Window Programming!
윈도우 프로시저
WinProc( )
메시지 루프
화면에 표시
WM_LBUTTONDOWN
처리 루틴 실행



윈도우 프로시저가 호출된다.
→ 처리 방법을 프로그램으로 작성

첫 번째 방법 : Win32 SDK 를 이용

Win32 SDK

- 윈도우에서 애플리케이션을 개발할 때 필요한 c언어용 표준 라이브러리
- 윈도우 프로그래밍 개발 순서
  - 윈도우 클래스를 정의(초기화) → 운영체제에 등록
  - 프레임 윈도우를 생성 → 윈도우를 화면에 출력
  - 메시지 루프를 구동한다.
  - 윈도우 프로시저에서 메시지를 처리한다.

두 번째 방법 : MFC (Microsoft Foundation Class) 를 이용

MFC

- C++로 작성된 클래스 라이브러리
- 구조적으로 만들어져 있다

→ 윈도우를 만들 때 필요한 대부분의 코드가 자동으로 생성된다.

장점	프로그래머가 쉽게 윈도우 프로그램을 작성할 수 있다.
단점	자동으로 생성되기 때문에 전체적인 윈도우 프로그래밍을 이해하는데 어렵다.

윈도우 프로그래밍을 작성하는 방법

Win32 SDLK를 이용한 윈도우 프로그램 기본 구조 이해

MFC를 사용한 윈도우 프로그램 작성 시 도움이 된다.

4. SDK 윈도우 프로그램의 기본 구조

학습 순서

- 윈도우 프로그램에서 사용하는 함수
  - 윈도우 프로그래밍 개발 순서에 따른 함수
- 실습 : 간단한 SDK 윈도우 프로그램 만들기

윈도우 프로그램의 기본 구조

초기화하는 부분

- WinMain() 함수에서 담당
  - 윈도우 프로그램 시작 : WinMain() 함수에서 시작
  - 윈도우 프로그램 종료 : WinMain() 함수에서 종료

메시지를 처리하는 부분

- WndProc() 함수에서 담당

윈도우 프로그램의 기본 구조

WinMain( ) 함수

```
Int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpszCmdLine, int nCmdShow)
{
        ① 윈도우 클래스 생성
        ② 윈도우 클래스 등록
        ③ 프레임 윈도우 생성
        ④ 프레임 윈도우 화면에 표시
        ⑤ 메시지 큐로부터 메시지를 받아 해당 프로시저로 보냄
}
```

클래스

- C++ 의 클래스 개념이 아니다.
- 윈도우의 종류를 나타낸다. (윈도우의 특징 등을 정의)

WinMain( ) 함수

```
Int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpszCmdLine, int nCmdShow)
```

HINSTANCE hInstance

- 메모리에 로드 된 실행파일의 위치를 나타내는 주소값
- 실행 파일에 포함된 리소스(비트맵, 아이콘 등)에 접근할 때 사용된다.

HINSTANCE hPrevInstance

- 16비트 윈도우에서 사용되던 매개변수
- 항상 NULL
- Windows95 이후부터는 사용하지 않는다.

WinMain( ) 함수

```
Int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpszCmdLine, int nCmdShow)
```

LPTSTR lpszCmdLine

- 프로그램이 실행될 때 명령행 인자를 전달하는 매개 변수
- 실행 파일의 경로 등을 나타내는 문자열 포인터

int nCmdShow

- 프로그램이 시작할 때 보여줄 윈도우 모양
- 최대화, 최소화, 보통 상태로 보여줄 것인지를 결정해주는 값

윈도우 Procedure

- 윈도우 시스템에서 들어온 메시지를 처리한다.
- 함수명 뒤에 Proc가 붙는다.
- 프로시저는 여러 개가 될 수 있다.
  - 윈도우 클래스마다 수행되는 프로시저가 다르다.

WndProc( ) 함수

메시지를 처리한다.

- 윈도우 시스템에서 들어온 메시지를 switch 문을 이용하여 처리

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message) {
        해당 메시지에 대한 처리
    }
}
```

WndProc( ) 함수

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

LRESULT

- 결과 값을 저장하는 32비트 자료형

CALLBACK 함수

어떤 동작에 의해 시스템에서 자동으로 호출되는 함수

- 메시지에 의해 자동으로 실행된다.
- WndProc( ) 함수
  - WinMain( ) 함수에서 직접 호출하지 않는다.
  - CALLBACK 함수로 자동으로 실행된다.
  - WinMain( ) 함수의 while 메시지 루프에 의해 감추어진 상태로 실행된다.
    - 메시지 루프의 DispatchMessage( ) 함수가 실제로 호출한다.

WndProc( ) 함수

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

- HWND hwnd
  - 윈도우의 핸들
- UINT message
  - WinMain( ) 함수에서 보내주는 메시지
- WPARAM wParam, LPARAM lParam
  - 메시지와 함께 필요한 정보가 들어오는 매개변수

5. 간단한 SDK 윈도우 프로그램 만들기

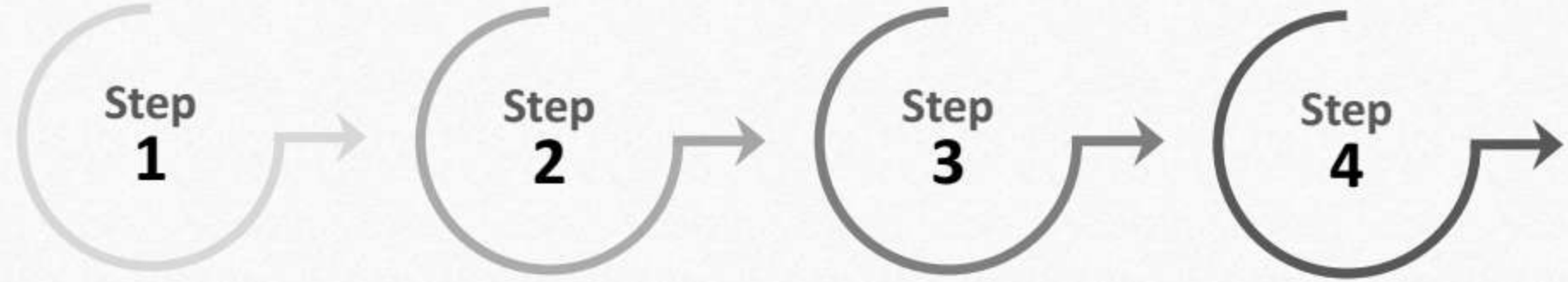
실습 목적

- 윈도우 프로그램의 기본 구조와 동작 원리에 대한 이해
- Win32 SDK를 이용한다
  - MFC를 사용한 윈도우 프로그램의 동작에 대한 기본 지식 습득

실습 내용

- 화면에 윈도우를 생성
- 텍스트 출력
  - "I love Window Programming!"
- 키보드와 마우스 이벤트에 대한 메시지를 출력
- Win32 SDK 기반의 프로그램 작성

실습 1-1



Win32
SDK 프로젝트를
만든다

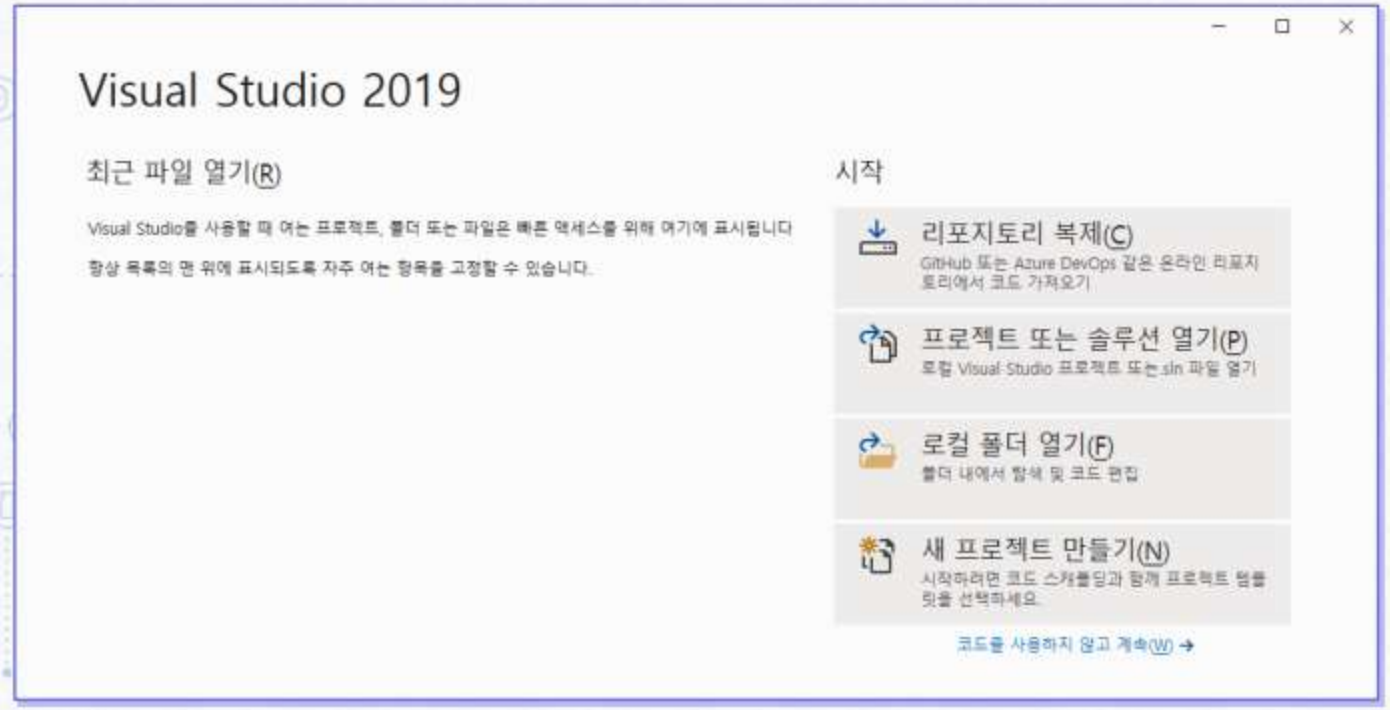
프로젝트에
소스파일을
삽입한다

Practice1_1.cpp
파일에
소스 코드를
입력한다

프로그램을
실행한다

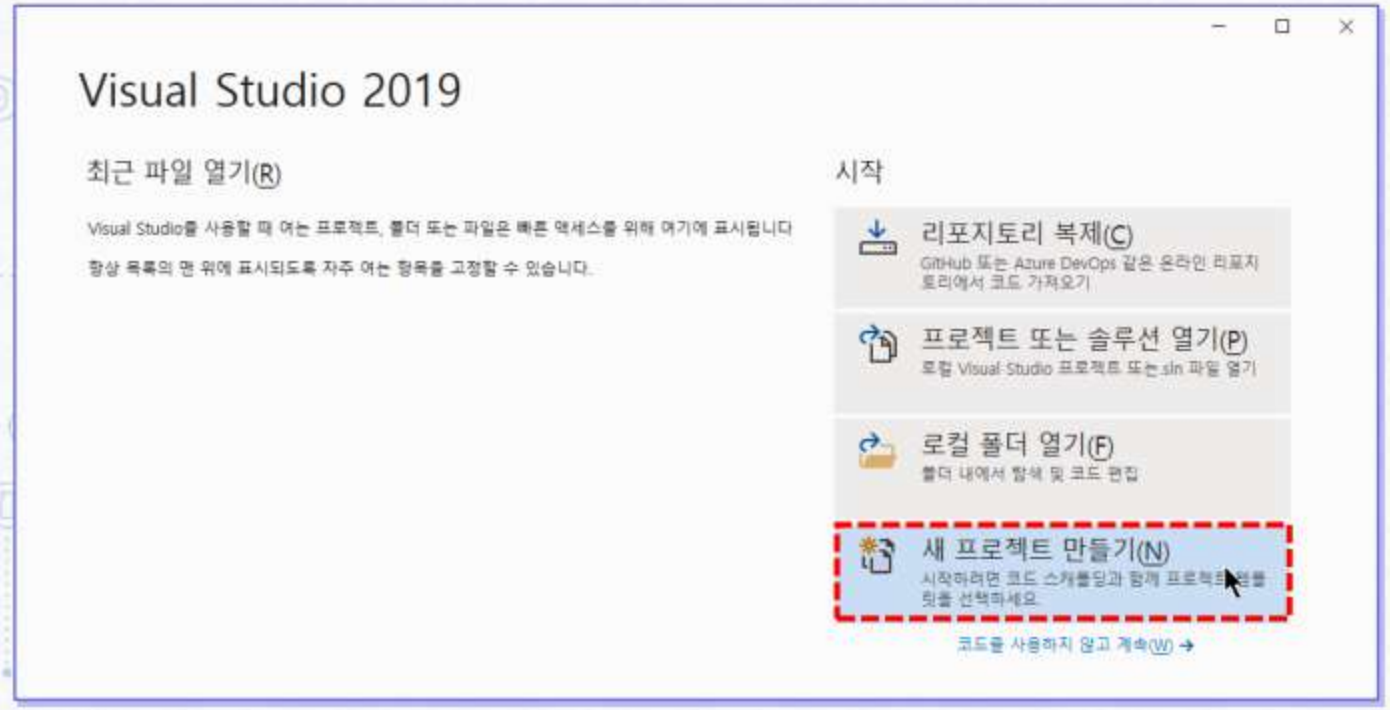
Step 1 : Win32 SDK 프로젝트를 만든다.

Visual Studio 2019 실행



Step 1 : Win32 SDK 프로젝트를 만든다.

새 프로젝트 만들기

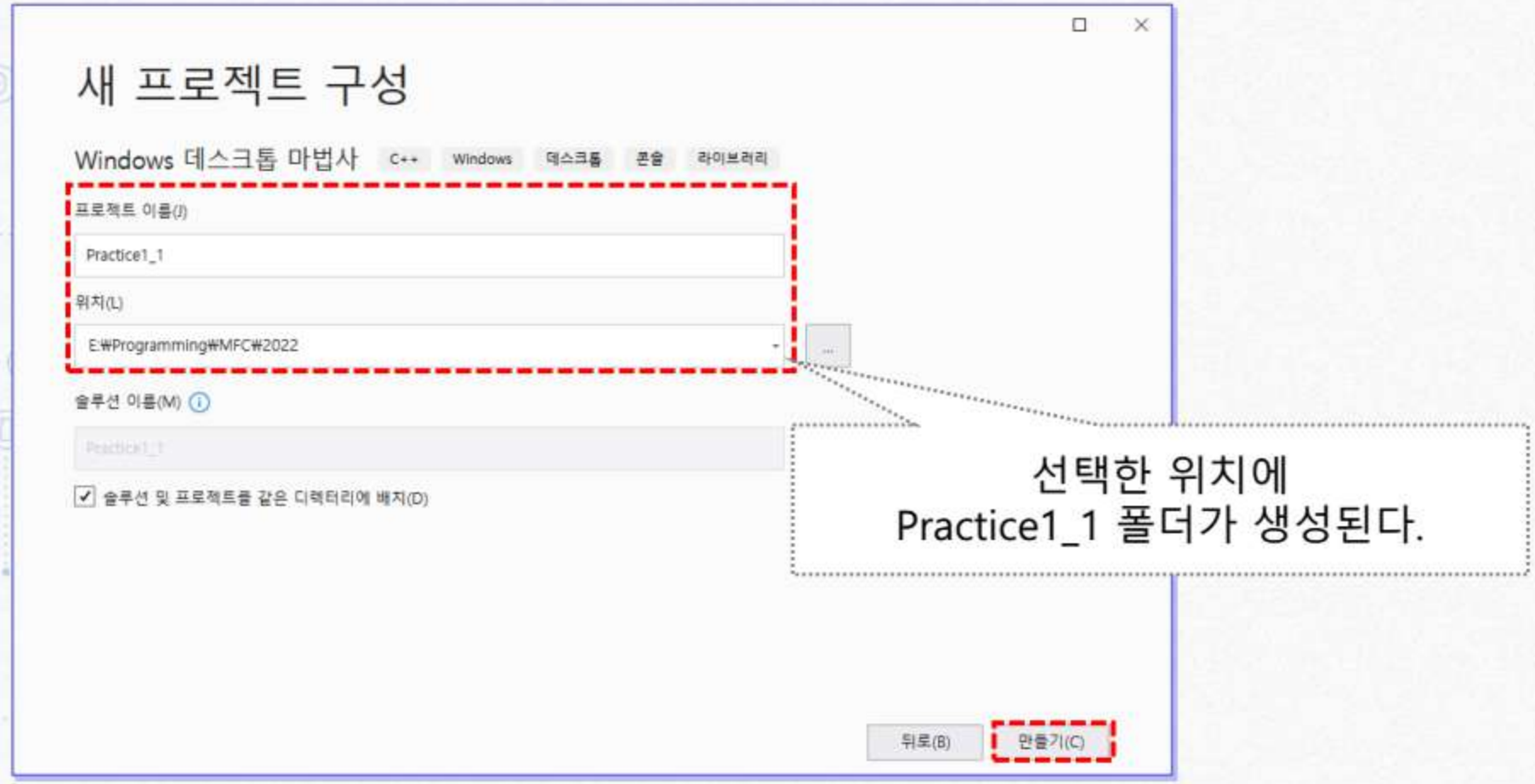


Step 1 : Win32 SDK 프로젝트를 만든다.

Windows 데스크톱 마법사 **선택** → 다음

Step 1 : Win32 SDK 프로젝트를 만든다.

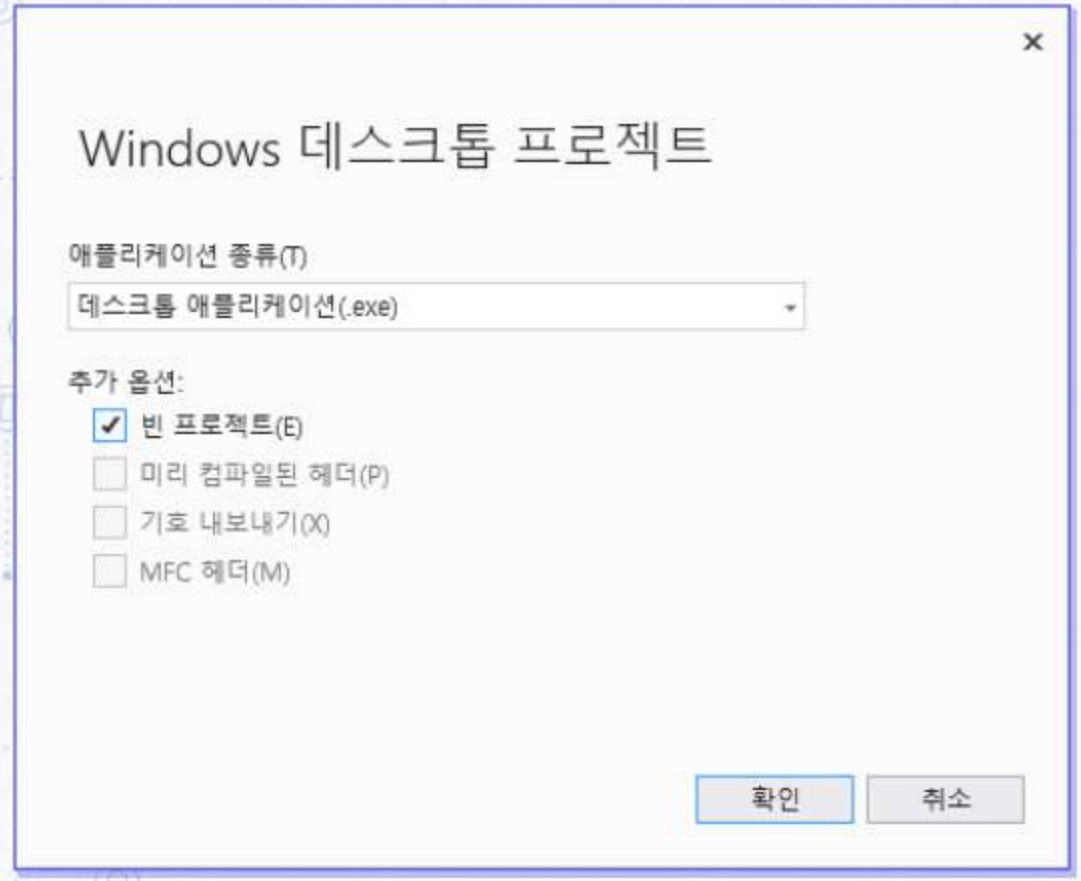
새 프로젝트 구성요소 입력



Step 1 : Win32 SDK 프로젝트를 만든다.

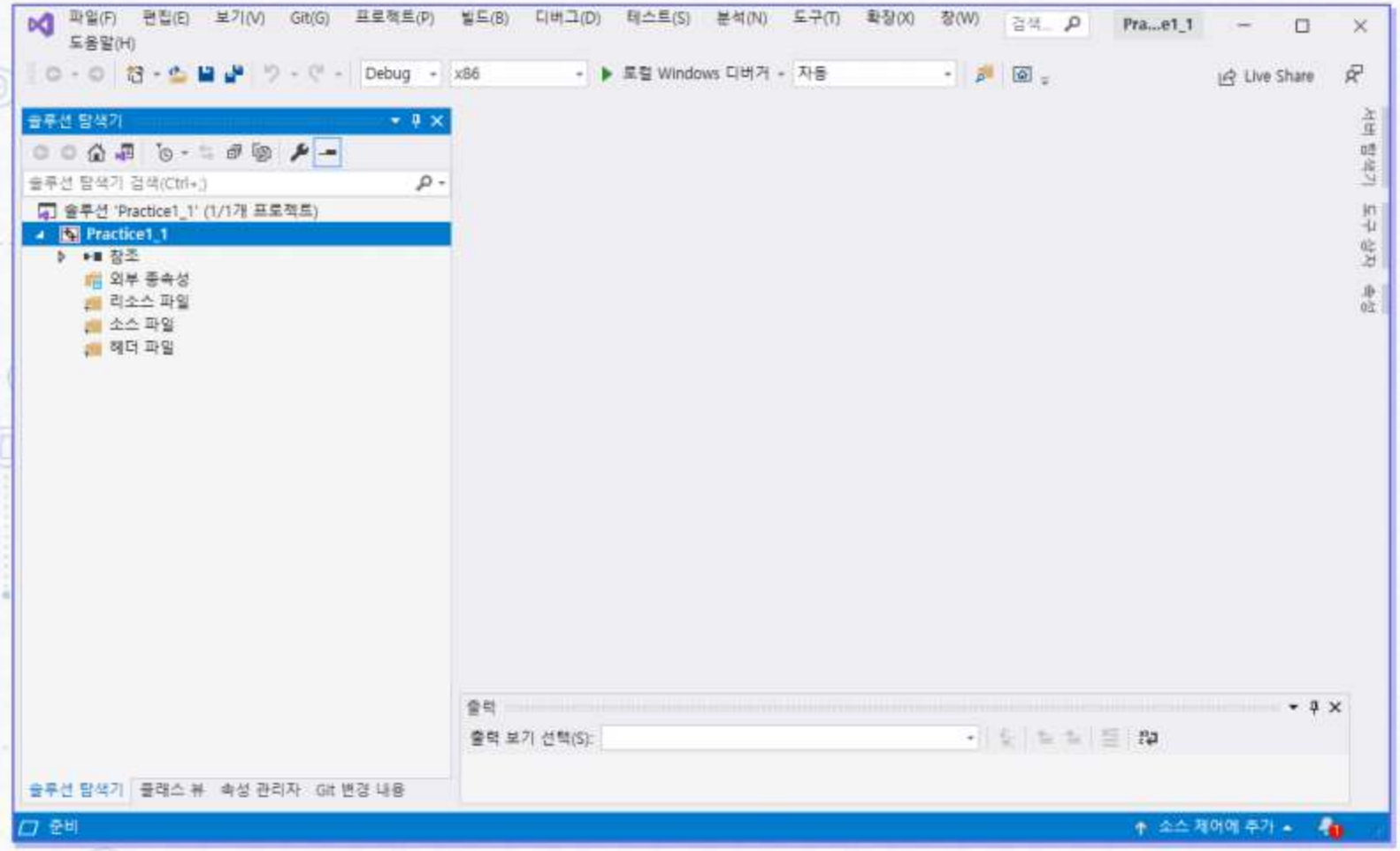
애플리케이션 종류

- 데스크톱 애플리케이션 선택



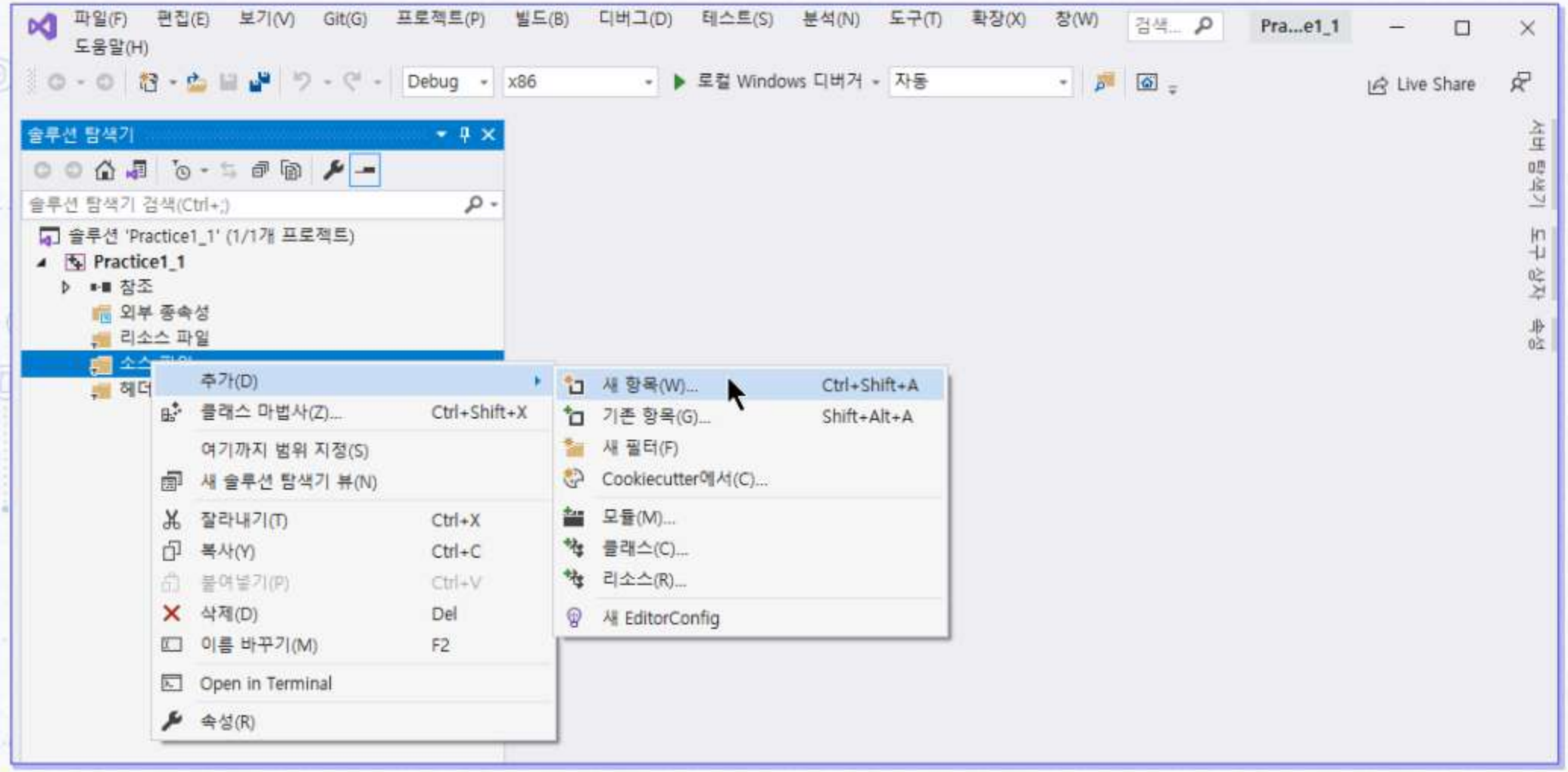
Step 1 : Win32 SDK 프로젝트를 만든다.

생성된 Practice1_1 프로젝트



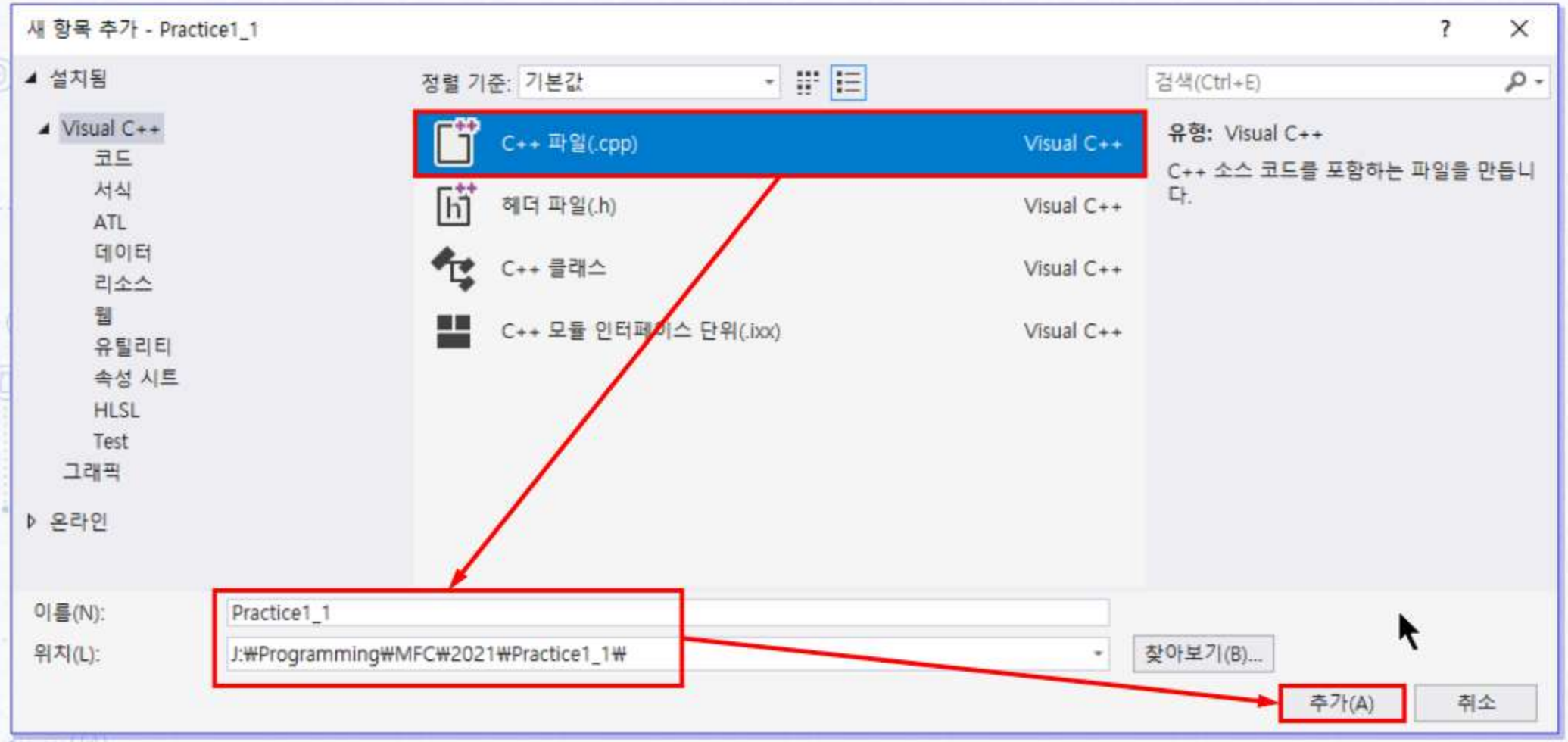
Step 2 : 프로젝트에 소스파일을 삽입한다.

솔루션 탐색기 → 소스파일 선택 → 마우스 우클릭 → 추가 → 새 항목



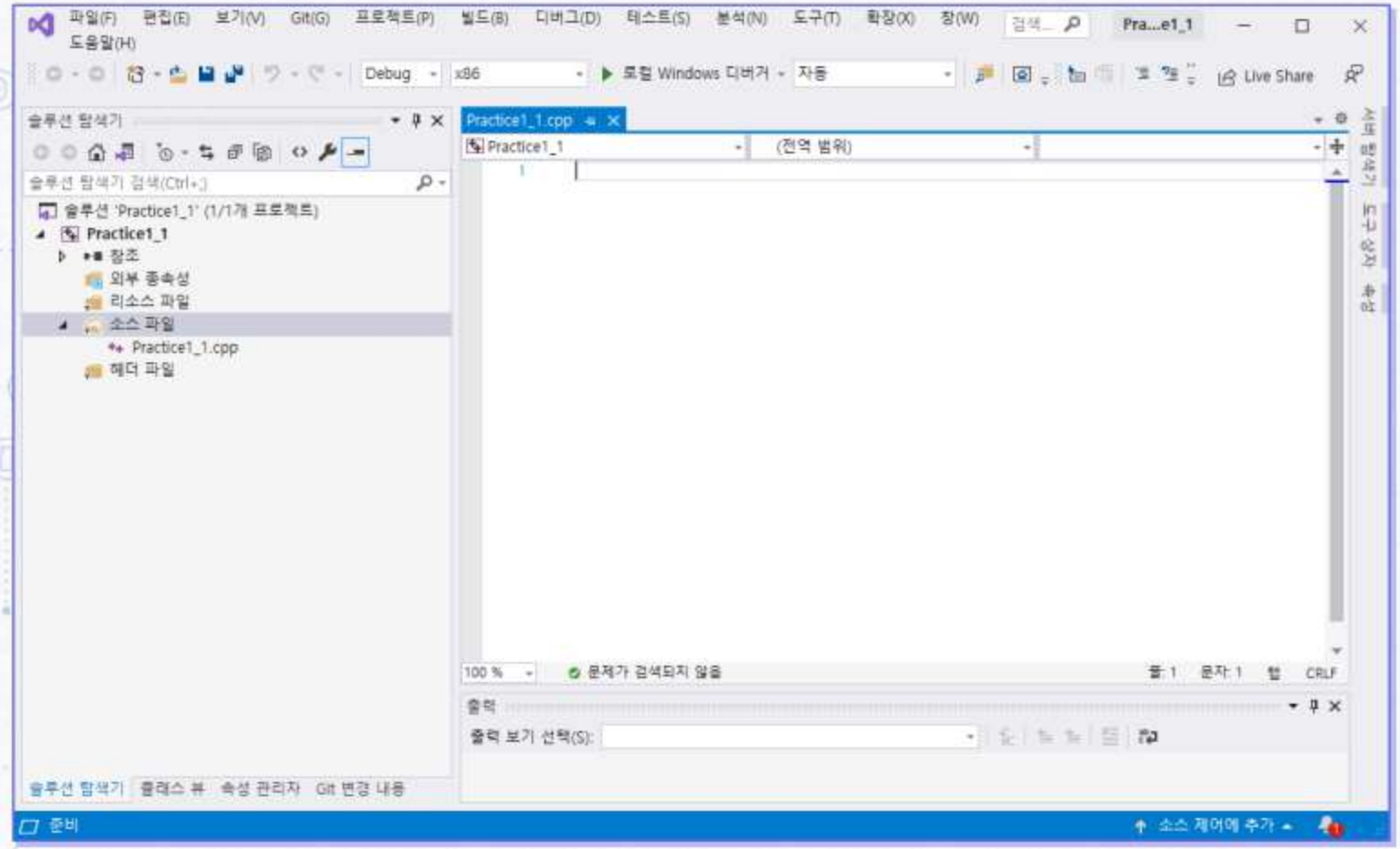
Step 2 : 프로젝트에 소스파일을 삽입한다.

C++ 파일 선택 → 이름 : Practice1_1 → 추가



Step 2 : 프로젝트에 소스파일을 삽입한다.

프로젝트에 Practice1_1.cpp 파일 생성



Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다.

구현 기능

- 윈도우 출력
- 윈도우에 "I love Window Programming!" 문자열 출력
- 키보드가 눌렸을 때 : "키보드가 눌러졌습니다." 문자열 출력
- 키보드가 떼어졌을 때 : "키보드가 떼어졌습니다." 문자열 출력
- 마우스 왼쪽 더블 클릭 : "마우스 더블 클릭!" 문자열을 가진 메시지 박스 출력

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다.

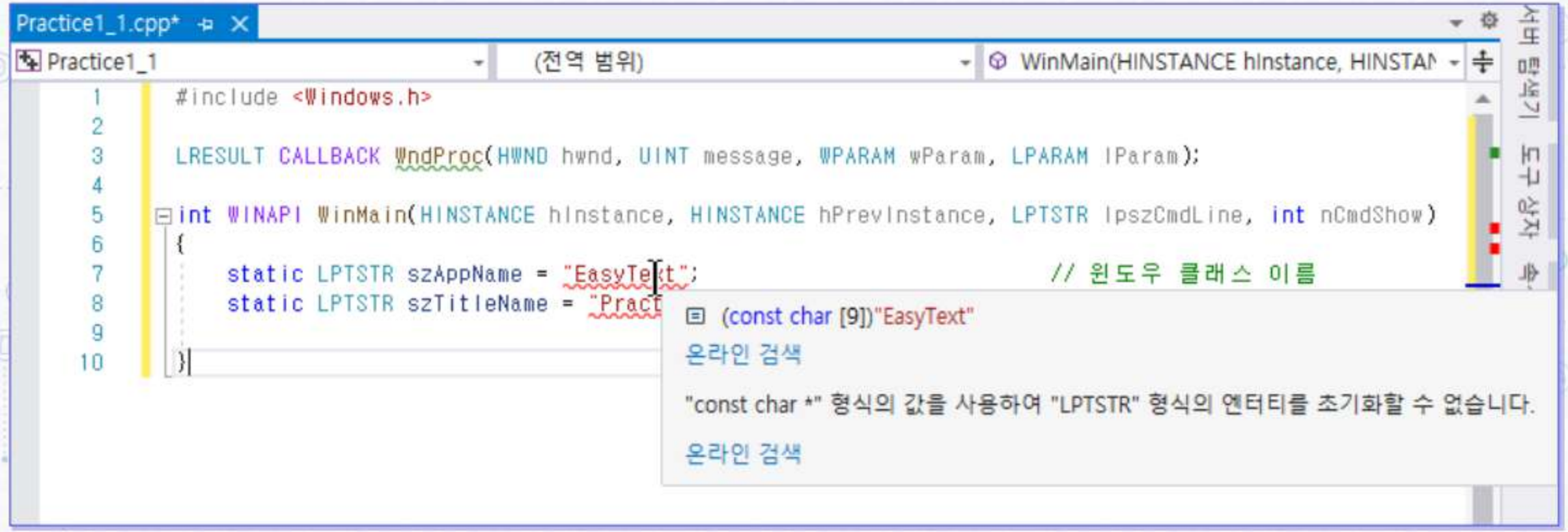
```
#include <Windows.h>

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

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpszCmdLine, int nCmdShow)
{
    static const LPTSTR szAppName = "EasyText";          // 윈도우 클래스 이름
    static const LPTSTR szTitleName = "Practice1-1";     // 타이틀 바에 출력될 문자열
}
```

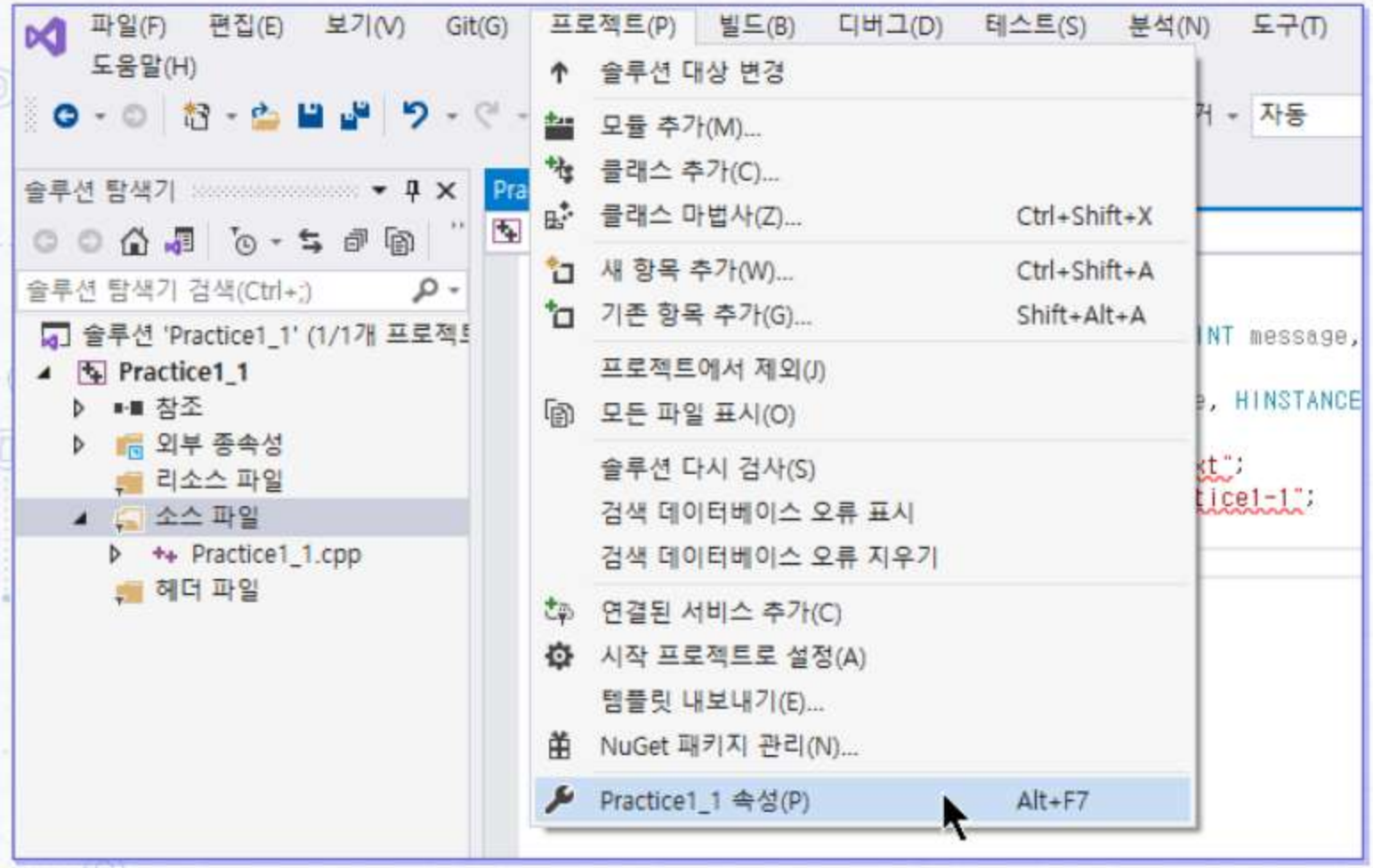
Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다.

입력 시 오류 발생



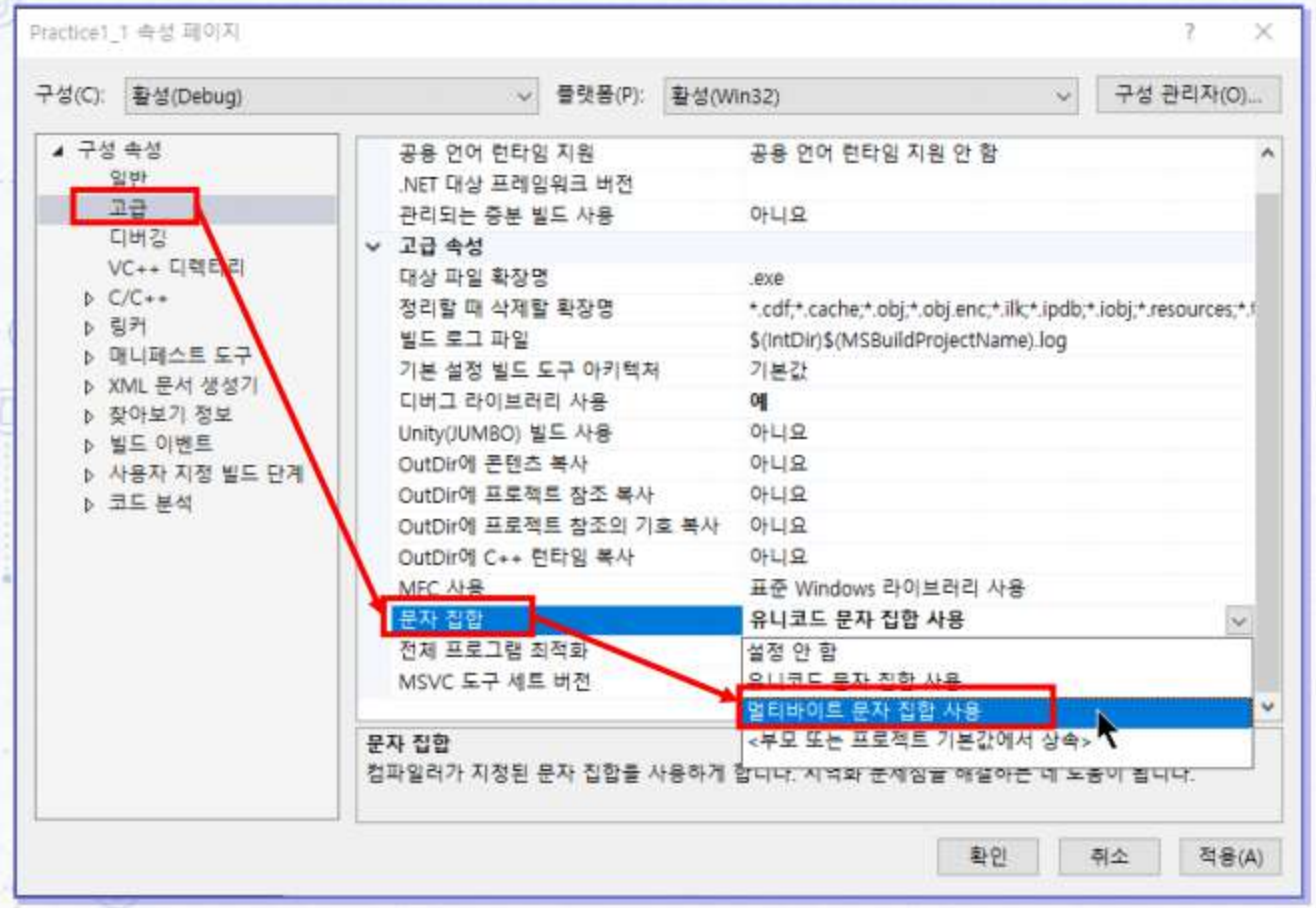
Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다.

입력 시 오류 발생 : 프로젝트 → Practice1_1 속성



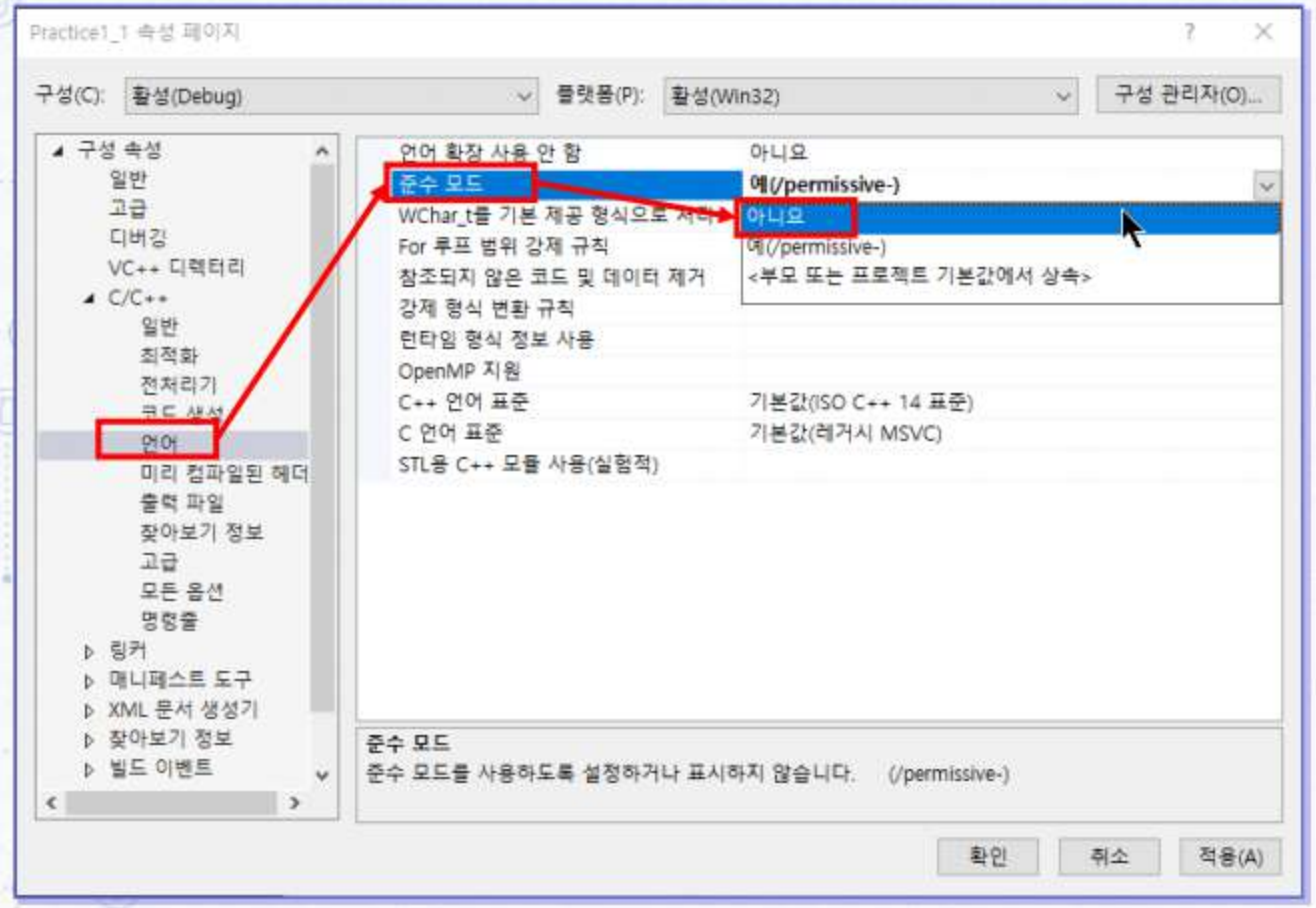
Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다.

- 입력 시 오류 발생 : 속성 페이지 → 구성 속성 → 고급
  - 문자 집합 > 유니코드 문자 집합 사용 → 멀티바이트 문자 집합 사용



Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다.

- 입력 시 오류 발생 : 속성 페이지 → 구성 속성 → C/C++ → 언어
  - 준수 모드 → 예 → 아니오



Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다.

Practice1_1.cpp*

Practice1_1 | (전역 범위) | WinMain(HINSTANCE hInstance, HINSTAN

```
1   #include <Windows.h>
2
3   LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);
4
5   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpszCmdLine, int nCmdShow)
6   {
7       static LPTSTR szAppName = "EasyText";                    // 윈도우 클래스 이름
8       static LPTSTR szTitleName = "Practice1-1";               // 타이틀 바에 출력될 문자열
9
10  }
```

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다. (WinMain 함수 – 1/4)

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpszCmdLine, int nCmdShow)
{
    static LPTSTR szAppName = "EasyText";                    // 윈도우 클래스 이름
    static LPTSTR szTitleName = "Practice1-1";               // 타이틀 바에 출력될 문자열

    HWND        hwnd;                                        // 윈도우 핸들
    MSG         msg;                                         // 메시지 구조체
    WNDCLASSEX  WndClass;                                    // 윈도우 클래스 구조체
```

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다. (WinMain 함수 – 2/4)

```
// ① 윈도우 클래스 구조체 WndClass에 값을 채워 윈도우 클래스를 등록한다.
WndClass.cbSize = sizeof(WNDCLASSEX);                          // 구조체 크기
WndClass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;         // 클래스 스타일
WndClass.lpfnWndProc = WndProc;                                // 윈도우 프로시저
WndClass.cbClsExtra = 0;                                       // 윈도우클래스 데이터영역
WndClass.cbWndExtra = 0;                                       // 윈도우의 데이터영역
WndClass.hInstance = hInstance;                                // 인스턴스 핸들
WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);              // 아이콘 핸들
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);                // 커서 핸들
WndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);  // 배경 브러시 핸들
WndClass.lpszMenuName = NULL;                                  // 메뉴 이름
WndClass.lpszClassName = szAppName;                            // 윈도우 클래스 이름
WndClass.hIconSm = 0;                                          // 기본적인 작은 아이콘

// 윈도우 클래스를 등록한다.
RegisterClassEx(&WndClass);
```

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다. (WinMain 함수 – 1, 2/4)

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpszCmdLine, int nCmdShow)
{
    static LPTSTR szAppName = "EasyText";                                   // 윈도우 클래스 이름
    static LPTSTR szTitleName = "Practice1-1";                              // 타이틀 바에 출력될 문자열

    HWND        hwnd;                                                       // 윈도우 핸들
    MSG         msg;                                                        // 메시지 구조체
    WNDCLASSEX  WndClass;                                                   // 윈도우 클래스 구조체

    // ① 윈도우 클래스 구조체 WndClass에 값을 채워 윈도우 클래스를 등록한다.
    WndClass.cbSize = sizeof(WNDCLASSEX);                                   // 구조체 크기
    WndClass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;                  // 클래스 스타일
    WndClass.lpfnWndProc = WndProc;                                         // 윈도우 프로시저
    WndClass.cbClsExtra = 0;                                                // 윈도우클래스 데이터영역
    WndClass.cbWndExtra = 0;                                                // 윈도우의 데이터영역
    WndClass.hInstance = hInstance;                                         // 인스턴스 핸들
    WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);                       // 아이콘 핸들
    WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);                         // 커서 핸들
    WndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);           // 배경 브러시 핸들
    WndClass.lpszMenuName = NULL;                                           // 메뉴 이름
    WndClass.lpszClassName = szAppName;                                     // 윈도우 클래스 이름
    WndClass.hIconSm = 0;                                                   // 기본적인 작은 아이콘

    // 윈도우 클래스를 등록한다.
    RegisterClassEx(&WndClass);
```

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다. (WinMain 함수 – 3/4)

```
// ② 프레임 윈도우를 생성한다.
hwnd = CreateWindow(                                    // 윈도우 생성 API 함수
    szAppName,                                          // 등록된 윈도우 클래스 이름
    szTitleName,                                        // 타이틀 바에 출력될 문자열
    WS_OVERLAPPEDWINDOW,                                // 윈도우 스타일
    CW_USEDEFAULT,                                      // 윈도우 좌측 상단의 x좌표
    CW_USEDEFAULT,                                      // 윈도우 좌측 상단의 y좌표
    CW_USEDEFAULT,                                      // 윈도우의 너비
    CW_USEDEFAULT,                                      // 윈도우의 높이
    NULL,                                               // 부모 윈도우의 핸들
    NULL,                                               // 메뉴 또는 자식 윈도우의 핸들
    hInstance,                                          // 애플리케이션 인스턴스 핸들
    NULL                                                // 윈도우 생성 데이터의 주소
);

// 프레임 윈도우를 화면에 표시한다.
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
```

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다. (WinMain 함수 – 4/4)

```
    // ③ 메시지 큐로부터 메시지를 받아와 메시지를 해당 윈도우 프로시저로 보낸다.
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);                    // 메시지를 번역한다.
        DispatchMessage(&msg);                     // 메시지를 해당 윈도우 프로시저로 보낸다.
    }
    return msg.wParam;
}
```

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다. (WndProc 함수 – 1/2)

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;                            // 디바이스 컨텍스트
    RECT rect;                          // RECT 구조체
    PAINTSTRUCT ps;                     // 페인트 구조체
    LPTSTR szMsg1 = "I love Window Programming!";                   // 윈도우에 출력될 문자열
    LPTSTR szMsg2 = "키보드가 눌러졌습니다.";                        // 키보드를 눌렀을 때 출력될 문자열
    LPTSTR szMsg3 = "키보드가 떼어졌습니다.";                        // 키보드를 떼었을 때 출력될 문자열

    // ① 커널에서 들어온 메시지를 switch문을 이용하여 처리
    switch (message)
    {
        case WM_CREATE:                                             // 윈도우가 처음 생성 메시지가 온 경우
            break;
        case WM_PAINT:                                              // 화면에 출력 메시지가 온 경우
            hdc = BeginPaint(hwnd, &ps);
            TextOut(hdc, 10, 10, szMsg1, strlen(szMsg1));           // 윈도우에 문자열을 출력
            EndPaint(hwnd, &ps);
            break;
        case WM_KEYDOWN:                                            // 키보드 버튼이 눌린 경우
            hdc = GetDC(hwnd);
            GetClientRect(hwnd, &rect);
            DrawText(hdc, szMsg2, strlen(szMsg2), &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
            ReleaseDC(hwnd, hdc);
            break;
```

Step 3 : Practice1_1.cpp 파일에 소스 코드를 입력한다. (WndProc 함수 – 2/2)

```
        case WM_KEYUP:                                          // 키보드 버튼이 떼어진 경우
            hdc = GetDC(hwnd);
            GetClientRect(hwnd, &rect);
            DrawText(hdc, szMsg3, strlen(szMsg3), &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
            ReleaseDC(hwnd, hdc);
            break;
        case WM_LBUTTONDBLCLK:                                  // 왼쪽 마우스를 더블 클릭한 경우
            MessageBox(hwnd, "마우스 더블 클릭!", "마우스메시지", MB_OK | MB_ICONASTERISK);
            break;
        case WM_DESTROY:                                        // 프로그램 종료 메시지가 온 경우
            PostQuitMessage(0);
            break;
        default:                                                // 그 외의 메시지가 온 경우
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;
}
```

Step 4 : 프로그램을 실행한다.

실행 방법

- 컴파일/링크
  - F7
  - [빌드] > [솔루션 빌드]
- 실행
  - Ctrl + F5
  - [디버그] > [디버그 하지 않고 시작]

Step 4 : 프로그램을 실행한다.

실행

- 초기 화면

Step 4 : 프로그램을 실행한다.

실행

- 마우스 좌 더블 클릭