1장 윈도우 프로그래밍 기초

2024년도 1학기 윈도우 프로그래밍

1장 학습 목표

- **학습 목표**
  - 윈도우 특징을 이해할 수 있다.
  - 윈도우 프로그래밍 개념을 익히고 요소들을 이해할 수 있다.
  - 윈도우 프로그램이 어떻게 구성되고 실행되는지 이해할 수 있다.

- **내용**
  - 윈도우 특징
  - 메시지 (이벤트) 의미
  - 윈도우 프로그램 구성
  - 윈도우 메인 함수와 윈도우 프로시저 함수

우리가 그 동안 해왔던 많은 게임들

대부분
운영체제 윈도우 기반의
게임들

2차원 또는 3차원 게임들

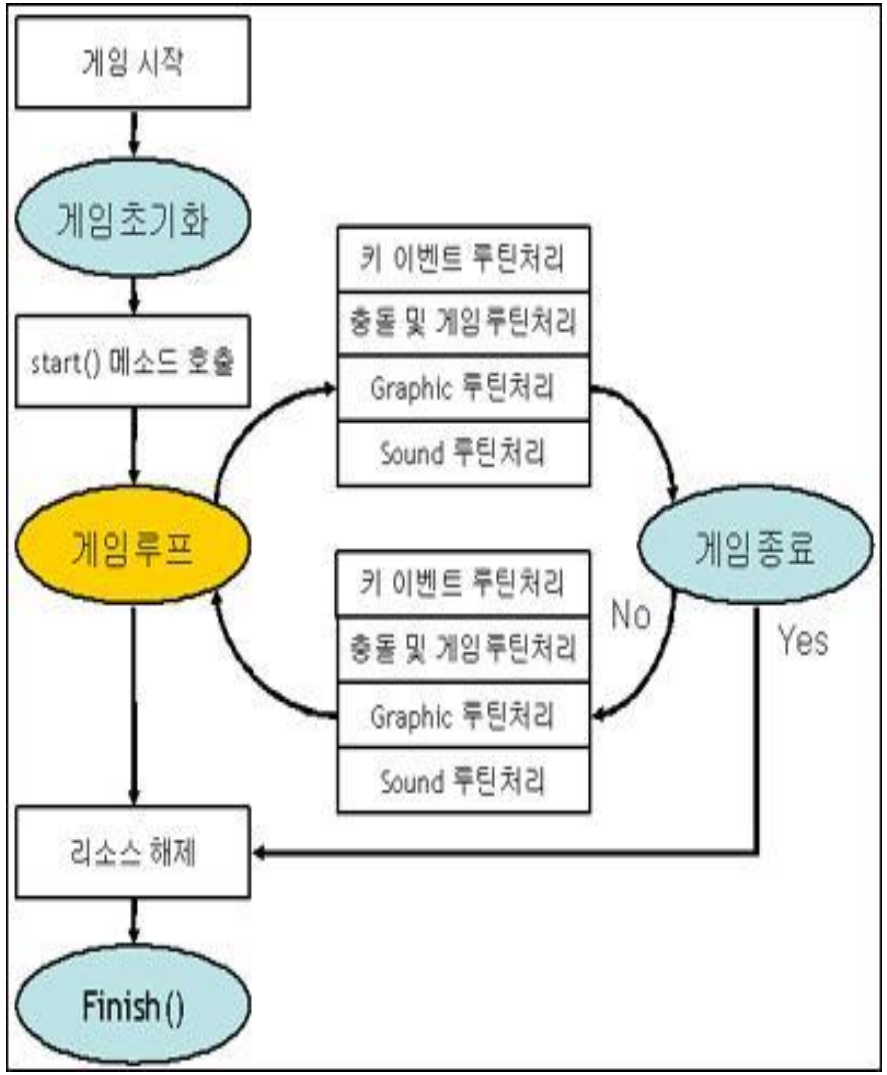
키보드나 마우스
이용해 게임 진행

스프라이트나 모델링 데이터를
이용한 애니메이션

버튼이나 스크롤 등의
컨트롤사용

그 게임들은

- 대개 아래의 flow chart에 따라 진행된다.



- 윈도우 띄우기
  - 메뉴, 단축키 사용
- 그래픽 처리하기
  - 캐릭터 등의 이미지
  - 애니메이션
- 키보드나 마우스 입력 받기
- 규칙에 의해 게임 진행하기

- 필요한 경우
- 다중 윈도우 띄우기
  - 윈도우 분할하기
- 다양한 컨트롤 사용하기
  - 버튼이나 선택 컨트롤 등
- 파일에 데이터 저장하기
  - 데이터 기록하고 읽어오기

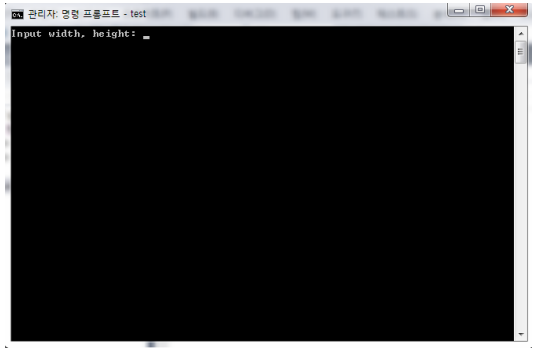
우리가 이번 학기에 배울 내용들

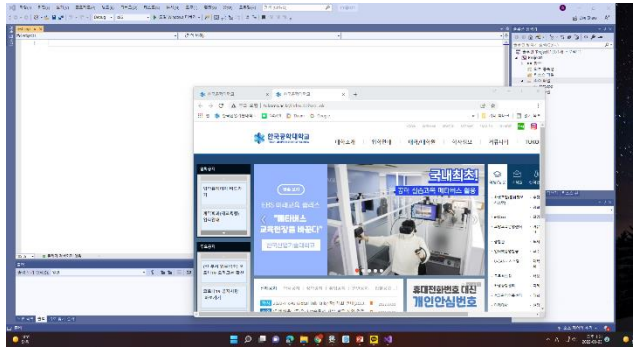
- 윈도우 띄우기
- 출력하기: 디바이스 컨텍스트
- 입력받기: 마우스, 키보드
- 그래픽 처리하기: 비트맵
- 애니메이션 만들기: 타이머
- 컨트롤 만들기: 다중 윈도우
- 데이터 저장: 파일 입출력

운영체제: DOS, Window

- **도스(DOS, Disk Operating System)는**
  - 문자기반의 운영체제
  - 절차적 프로그램
  - 프로그램의 실행 흐름이 **프로그래머가 기술한 코드 순서대로 순차적으로 진행**
  - 사용자가 키보드로부터 문자 명령어나 파일명 등을 입력하여 프로그램을 제어

- **윈도우는**
  - 그래픽 인터페이스 기반의 운영체제
  - 윈도우 응용 프로그램은 순차적으로 실행되지 않는다.
  - 프로그램의 실행 흐름을 **프로그래머 혼자 결정하지 않고 윈도우OS와 상호작용하면서 처리**
  - 이벤트 (메시지)를 기반으로 구동되는 방식
  - 어떤 메시지를 받는가에 따라 코드의 실행 순서가 달라지고, 메시지에 어떻게 반응하는가에 따라 동작이 달라진다.





윈도우 운영체제

- **Window 1.0과 3.1**
  - 1985년 11월 마이크로소프트사에서는 MS-DOS 를 기반으로 한 windows 1.0 버전 발표
    - 그래픽 유저 인터페이스 도입
    - 255KB 메모리 지원, 256 컬러 표시
  - 1990년 5월 windows3.0, 1992년 4월 windows 3.1 발표하면서 본격적인 윈도우 시대 도래

- **현재**
  - Window 10
    - 2015년 소개
    - 32 비트 버전과 64비트 버전 모두 제공
  - Window 11
    - 2021년 배포
    - WinUI 3 개선: 새로운 모양의 윈도우 11창 환경 지원 (창의 둥근 모서리, 새로운 글꼴, 새로운 아이콘 모음 등)
    - 64비트 버전으로만 제공, 32비트 어플리케이션은 실행되고 작동됨
    - 윈도우 앱 SDK 를 차세대 윈도우 SDK로 공식화 함

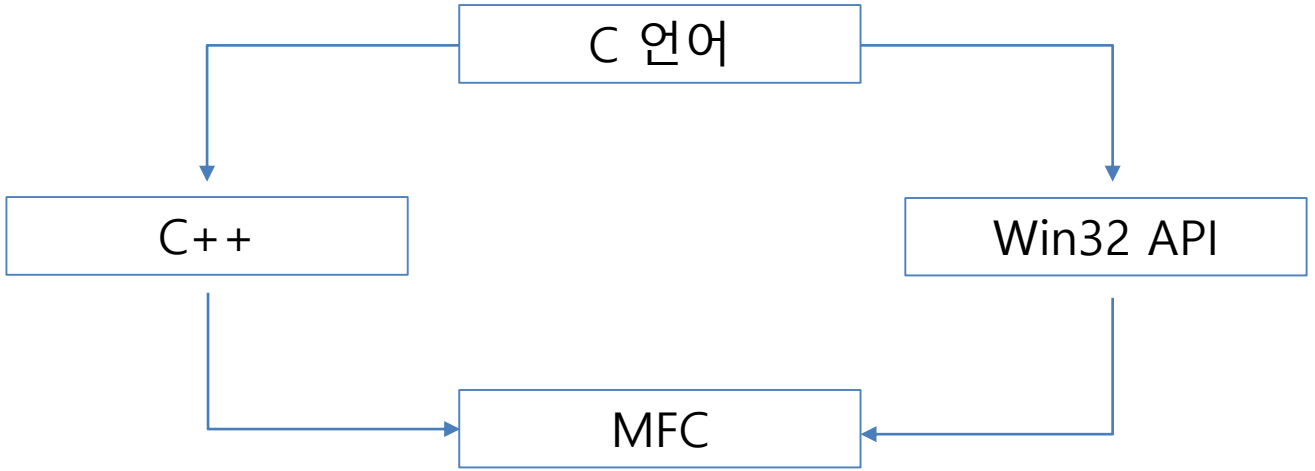
윈도우 운영체제의 특징

- **윈도우 운영 체제의 특징**
  - 그래픽 사용자 인터페이스 (GUI) 기반의 운영체제
    - 픽셀 단위의 그래픽 기반 운영 체제
  - **이벤트 기반 (메시지 기반) 시스템**
    - 이벤트: 사용자가 GUI를 통해 응용 프로그램과 소통할 때 사용자에 의해 발생하는 사건
    - 메시지: 윈도우가 응용 프로그램에게 보내는 알림
    - 외부에서 발생한 일들을 윈도우 OS가 감지하여 해당 프로그램에 메시지를 전달한다.
    - 프로그램의 실행 순서는 프로그래머가 미리 의도한 대로가 아니라 실행 중에 사용자가 프로그램을 조작하는 순서, 즉 발생하는 메시지의 순서를 따른다.
  - 장치 독립적
    - 하드웨어 장치에 무관하게 프로그래밍할 수 있다.
    - Device Driver에 의해 주변 장치들을 제어, 관리한다.
    - 프로그래머는 어떤 하드웨어가 현재 시스템에 설치되어 있는지 신경 쓸 필요가 없다.
  - 리소스 분리
    - 코드 이외의 데이터가 분리되어 있다.
  - 멀티 태스킹 기반 프로그램
    - 한 번에 여러가지 일을 동시에 수행할 수 있는 방식
    - 응용 프로그램은 메모리, CPU, 디스크, 화면 등 프로그램과 자원을 공유
  - 일관성이 있다
    - 사용자가 프로그램에게 명령을 내리는 인터페이스 구성이 표준화되어 있다.

윈도우 프로그램 개발 학습 과정

- **C나 C++는 표준화 언어**
  - 윈도우, 유닉스, 리눅스와 같은 운영체제에서도 사용된다.

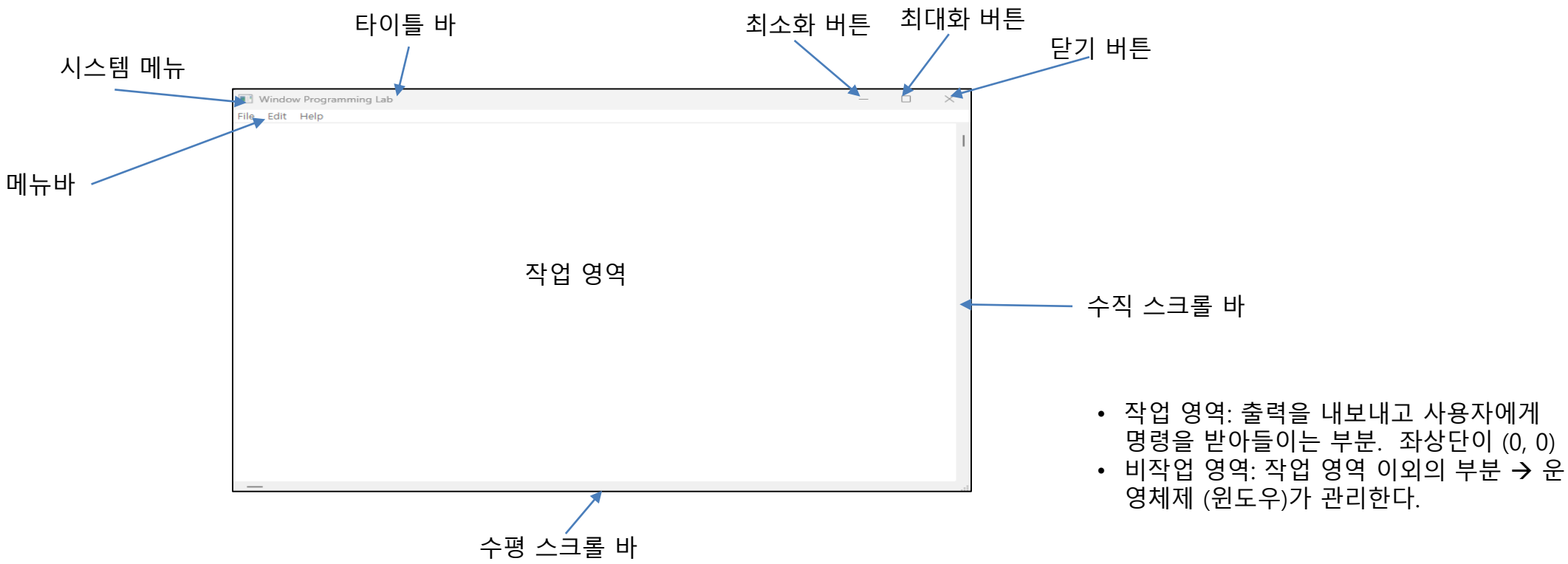
- **프로그램을 작성할 때는**
  - 프로그램이 실행되는 운영체제에서 제공하는 함수를 이용
  - MS 윈도우: C 언어와 Win32 API를 사용하여 윈도우 프로그램 개발
  - Win32 API: 윈도우 응용 프로그램에서 필요한 기능을 라이브러리 함수 호출 형태로 사용할 수 있게 한다.
  - MFC (Microsoft Foundation Class): 윈도우 운영체제에서 작동하는 GUI 프로그램을 C++을 사용하여 개발할 수 있도록 Win 32API와 C언어 함수들을 C++ 언어의 클래스화 한 라이브러리



윈도우

- **윈도우 정의**
  - 프로그램이 출력 결과를 내 보내고 사용자로부터 입력을 받아들이는 화면상의 사각 영역
    1) 화면상에 존재
    2) 모양은 직사각형 (높이와 폭이 있고 각 변끼리는 수직을 이룬다)
    3) 독립적으로 사용자와 상호작용 할 수 있다.

- **윈도우의 구성 요소**



- 작업 영역: 출력을 내보내고 사용자에게 명령을 받아들이는 부분. 좌상단이 (0, 0)
- 비작업 영역: 작업 영역 이외의 부분 → 운영체제 (윈도우)가 관리한다.

윈도우 프로그램 구성 요소

- **윈도우 프로그래밍의 구성 요소**
  - 메시지 (이벤트)
    - 윈도우에서 발생하는 모든 이벤트
    - 윈도우에서 이벤트가 발생하는지 확인하고 메시지로 만들어 처리하는 것은 운영체제의 역할
  - 메시지 큐
    - 운영체제가 만들어낸 메시지를 저장하는 곳
    - 먼저 들어온 메시지를 순차적으로 처리한다
  - 메시지 루프
    - 메시지 큐에 들어있는 메시지를 읽어서 처리를 위해 윈도우 프로시저로 전송하는 반복문 (루프)
    - 메시지에 따라서 알맞은 형태로 변환해서 윈도우 프로시저로 전송
  - 윈도우 프로시저
    - 메시지 루프에서 전송한 메시지를 받아 처리하는 함수
    - 개발자는 프로시저의 메시지 처리 방식을 정의해서 프로그램을 개발

윈도우 프로그램 구성 요소

- **메시지(Message) 와 이벤트(Event)**
  - 이벤트: 사용자가 키보드를 누르거나 마우스 버튼을 클릭할 때, 툴 바의 버튼을 누르거나 윈도우의 크기를 조절하는 등의 기계적인 조작에 의해 발생
  - 메시지: 어떤 이벤트가 발생하면 운영체제는 메시지를 만들어서 해당 윈도우가 속한 메시지 큐에 저장

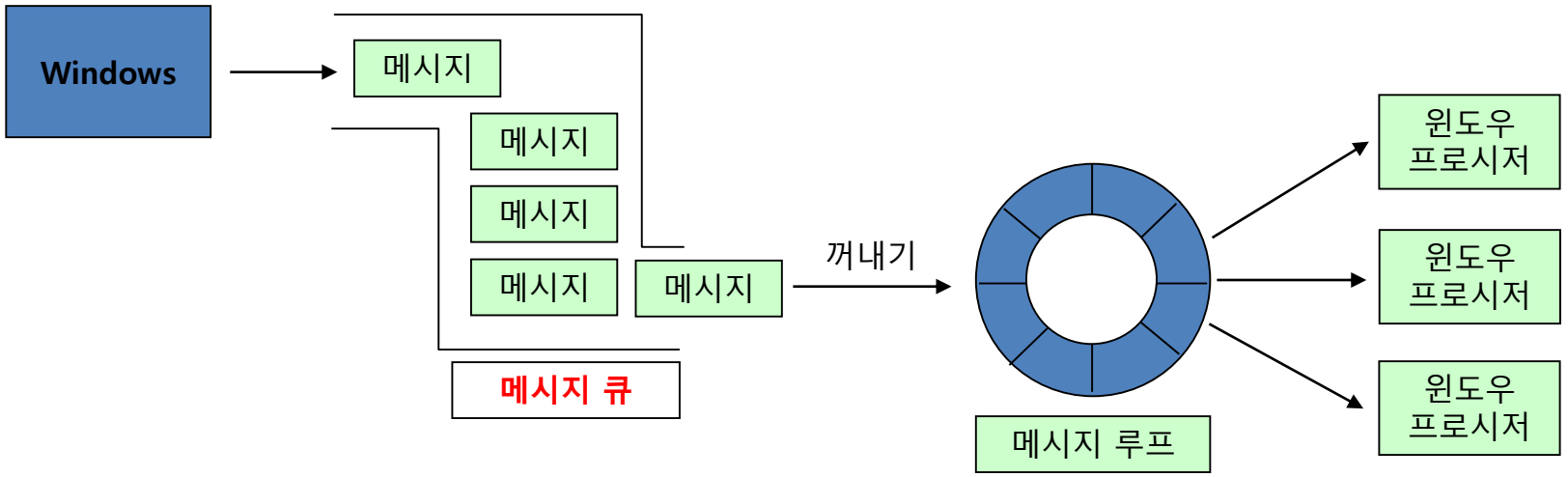
  - 이벤트가 발생하면 윈도우 OS는 이를 감지하여 해당 프로그램으로 메시지를 전달
  - 예) 마우스 누름(이벤트) → WM_LBUTTONDOWN 메시지로 변환
    - 윈도우 메시지 유형들은 모두 "WM_"로 시작

윈도우 메시지 유형	이벤트 (발생하는 상황)
WM_CREATE	윈도우가 생성될 때
WM_PAINT	윈도우가 다시 그려져야 할 때
WM_MOUSEMOVE	마우스 커서가 움직였을 때
WM_COMMAND	메뉴 등으로 명령을 내렸을 때
WM_LBUTTONDOWN	마우스 왼쪽 버튼이 눌렸을 때
WM_TIMER	설정된 타이머 시간이 되었을 때
WM_DESTROY	윈도우가 없어질 때

<여러 종류의 메시지들>

윈도우 프로그램 구성 요소

- **메시지 큐(Message Queue)**
  - 큐(Queue): FIFO (First In First Out) 타입의 저장 공간으로 먼저 들어온 메시지가 먼저 처리된다.
  - 사용자의 컴퓨터 조작에 의해 발생한 이벤트는 메시지 형태로 만들어져 윈도우 OS가 관리하는 메시지 큐라는 곳에 모이게 됨
  - 하나의 프로그램이 실행되면 하나의 메시지 큐가 할당됨



윈도우 프로그램 구성 요소

- **메시지 루프(Message Loop)**
  - 윈도우 OS가 프로그램에 전달한 메시지를 받아들여 분석하는 무한 루프
    - 사용자나 시스템 내부적인 동작에 의해 메시지가 발생하면
    - 프로그램에서는 메시지가 어떤 정보를 담고 있는가를 분석하여 어떤 루틴을 호출할 것인가를 결정
    - 이때, 메시지를 처리하는 부분을 메시지 루프라고 한다
  - 일반적인 메시지 루프 형태

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

  - 메시지 큐에서 메시지를 꺼내고 필요한 경우 형태를 바꾼 후 응용 프로그램 (윈도우 프로시저)으로 전달
  - GetMessage() 함수가 FALSE를 리턴 (프로그램을 종료하라는 WM_QUIT 메시지일 경우)할 때까지 메시지 큐로부터 메시지를 얻어와 처리

윈도우 프로그램 구성 요소

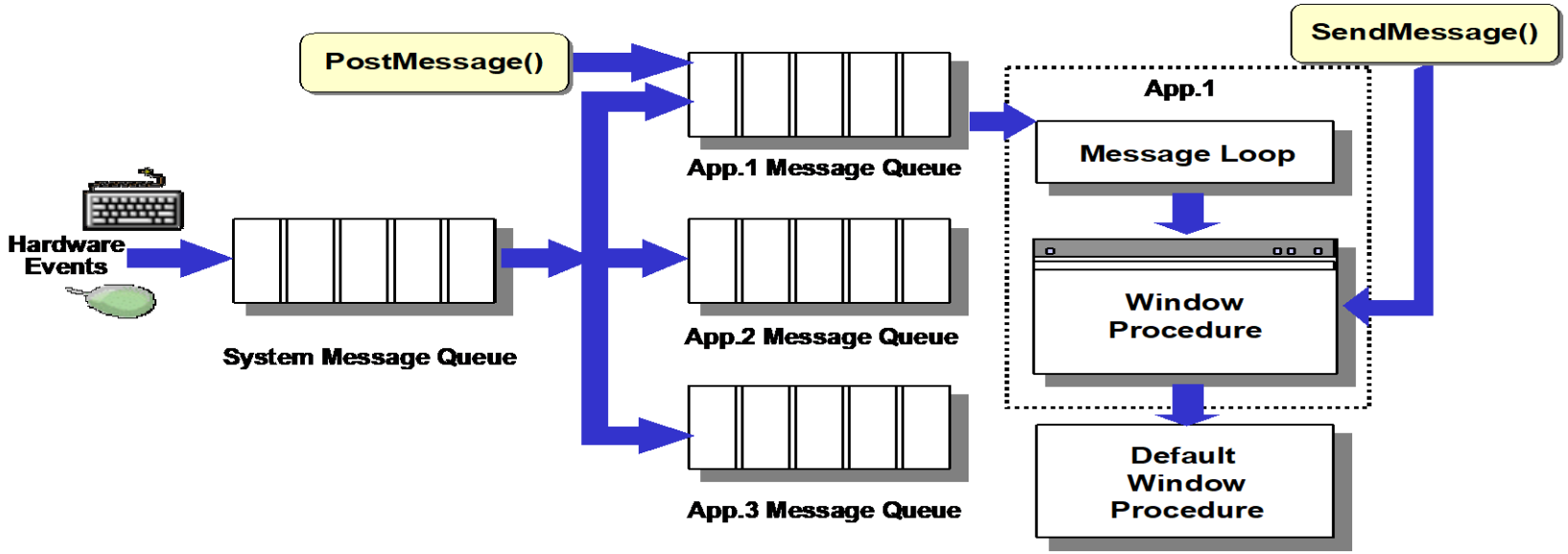
- **윈도우 프로시저(Window Procedure)**
  - 메시지 루프에서 해석한 메시지를 구체적으로 처리하는 기능을 하는 메시지 처리 함수
  - 이 함수는 WinMain 에서 호출하는 것이 아니라 윈도우 OS가 호출하는 콜백 함수(Callback function)
    - **콜백 함수: 운영체제가 호출하는 함수**
  - 콜백 함수는 함수 앞에 키워드 CALLBACK을 붙인다.
  - 함수의 이름은 프로그래머가 마음대로 지정할 수 있다.
    - 대개 Window와 Procedure를 그대로, 또는 줄인 후 혼합하여 사용
  - 일반적인 윈도우 프로시저 함수 형태

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hDC;

    switch (iMessage) {
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;
    }
    return (DefWindowProc(hWnd, iMessage, wParam, lParam));
}
```

윈도우 프로그램 동작 방식

- **사용자가 키보드나 마우스를 조작했을 때**
  - 운영체제가 적절히 처리
    - 윈도우는 멀티태스킹을 지원
    - 운영체제가 입력을 받는다 → 운영체제는 사용자가 조작중인 애플리케이션에 입력값을 분배한다.

  - 즉, 사용자가 키보드를 입력 → 인터럽트 발생 → 윈도우가 처리 → 대상 애플리케이션에 통지
    - 윈도우 애플리케이션은 자신이 소유한 윈도우에 대해 **윈도우 프로시저 함수** 준비
    - **윈도우 메시지를 사용하여 통지**한다.
    - **윈도우에서는 이벤트가 발생했을 때마다 메시지 내용을 인자로 넘겨서 윈도우 프로시저를 호출한다.**
    - 윈도우 프로시저는 윈도우에서 다양한 이벤트가 발생했을 때 이를 처리한다.
      - 윈도우 프로시저: 메시지를 구체적으로 처리하는 기능을 하는 소스 부분으로 OS가 호출하는 콜백 함수



윈도우 프로그램 동작 방식

- **윈도우 애플리케이션은 이벤트 반응형**
  - 이벤트 반응형: 사용자 조작으로 이벤트가 발생하고 그때마다 대응되는 처리를 하는 프로그램 형식
    - 운영체제인 윈도우가 장치를 감시하다가 이벤트가 발생했을 때 이벤트가 발생했음을 프로그램에 알리는 구조
    - 하드웨어적인 이벤트 감시는 하드웨어의 장치 드라이버의 역할
    - 장치 드라이버는 이벤트를 감지했을 때 일반적인 형태로 변한 뒤 이벤트 처리 행렬인 시스템 큐에 저장
    - 윈도우에서 애플리케이션 윈도우에 이벤트를 통지할 때: **윈도우 메시지**라고 하는 데이터 구조체를 이용

```
typedef struct tagMSG {
        HWND hwnd;                  //--- 윈도우 핸들
        UINT message;               //--- 메시지 id
        WPARAM wParam;              //--- 메시지 전달 인자 1: 32 (또는 64) 비트 LONG_PTR
        LPARAM lParam;              //--- 메시지 전달 인자 2: 32 (또는 64) 비트 UINT_PTR
        DWORD time;                 //--- 이벤트 발생 시각
        POINT pt;                   //--- 이벤트 발생 시 커서 위치
} MSG, *PMSG;
```

윈도우 API

- **윈도우 API (Windows Application Programming Interface)**
  - **운영체제인 윈도우의 기능을 애플리케이션에서 이용하기 위한 인터페이스**
  - 그 실체는 **C/C++ 나 비주얼 베이직 등의 다양한 언어/개발 툴에서 호출할 수 있는 수천 개의 함수 집합**
  - C/C++ 프로그램에서 운영체제와 응용 프로그램 사이의 정보 교환을 가능하게 한다.

  - 윈도우 API로 구현할 수 있는 작업
    - 메모리 관리, 입출력 명령, 프로세스와 스레드 생성, 동기화 함수들 등 대부분의 기본적인 윈도우 기능들
    - 그래픽 장치 인터페이스
    - 디스플레이나 프린터에 출력되는 원시적인 드로잉 함수들을 수행하는 역할
    - 창이나 메뉴 같은 윈도우 사용자 인터페이스의 표준 요소들을 생성하고 다룬다
    - 파일 오픈, 저장, 상태바 같은 다양한 종류의 윈도우 표준 컨트롤을 구현한다

윈도우 API

- **API는 DLL (Dynamic Link Library)안에 있다.**
  - Library: 소프트웨어 개발에서 자주 쓰고 기초적인 함수들을 중복 개발하는 것을 피하기 위해 표준화된 함수 및 데이터 타입을 만들어서 모아 놓은 것
    - Static Link Library(정적 링크): 컴파일 시점에 라이브러리가 링커에 의해 연결되어 실행 파일의 일부분이 된다.
    - Dynamic Link Library(동적 링크): 실행 파일에서 해당 라이브러리의 기능을 사용 시에만, 라이브러리 파일을 참조하여 기능을 호출한다. 정적 링크와는 다르게 컴파일 시점에 실행 파일에 함수를 복사하지 않고, 함수의 위치정보만 갖고 그 함수를 호출할 수 있게 한다.
  - API는 C언어로 기술된 응용 프로그램을 위한 함수들 모임으로 DLL로 저장되어 있다.
  - 윈도우 운영체제의 구성 모듈

모듈	파일명	기능
커널	KERNEL32.DLL	윈도우 OS의 핵심으로 메모리 관리, 파일 입출력, 프로그램의 로드와 실행 등 OS의 기본 기능 수행
GDI	GDI32.DLL	화면이나 프린터와 같은 출력 장치에 출력을 관리
사용자 인터페이스	USER32.DLL	윈도우, 다이얼로그, 메뉴, 커서, 아이콘 등과 같은 윈도우 기반의 사용자 인터페이스 객체들을 관리

윈도우 프로그램

- **일반적인 윈도우즈 프로그램은**
  - 코드: C/C++언어 소스부분과 관련 헤더,
  - 리소스: 리소스 스크립트 파일(.rc)과 리소스 파일 등으로 구성되어 있다.

구분	내용	확장자
코드	C/C++ 언어 소스 부분	*.c , *.cpp
	관련 헤더	*.h
리소스	리소스 스크립트 파일	*.rc
	리소스 파일	*.bmp, *.icn …

윈도우 프로그램의 형태

- **코드 부분은**
  - 한 개의 메인 함수 WinMain()함수와 한 개 이상의 윈도우 프로시저 함수로 구성된다.

```
#include  <windows.h>

int WINAPI WinMain (.....)
{
      윈도우 생성
      메시지 전송
 }
```

메인 부분: 메인 함수

WinMain 함수

- 윈도우 클래스 정의
- 윈도우 클래스 등록
- 윈도우 생성
- 윈도우 출력
- 메시지 루프
- **WinMain () 함수**

```
LRESULT CALLBACK WndProc (....)
{
      메시지에 따른 처리
}
```

메시지 처리 부분: 윈도우 프로시저 함수

Window Procedure 함수

- 메시지를 받아 약속된 반응을 보인다.
- **Window Procedure () 함수**

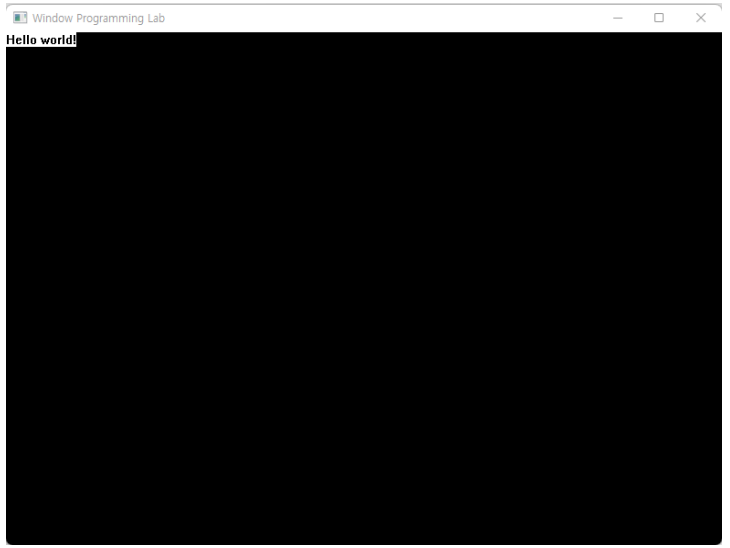
도스 기반 프로그램

```
#include <stdio.h>

void main ()
{
        printf ("Hello world₩n");
}
```

윈도우 기반 프로그램

```
#include <windows.h>                    //--- 윈도우 헤더 파일
#include <tchar.h>

HINSTANCE g_hInst;
LPCTSTR lpszClass = L"Window Class Name";
LPCTSTR lpszWindowName = L"Window Programming Lab";

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMessage,  WPARAM wParam, LPARAM lParam);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                          LPSTR lpszCmdParam, int nCmdShow)
{
    HWND hWnd;
    MSG Message;
    WndClassEX WndClass;
    g_hInst=hInstance;

    WndClass.cbSize = sizeof(WndClass);
    WndClass.style=CS_HREDRAW | CS_VREDRAW;
    WndClass.lpfnWndProc=(WNDPROC)WndProc;
    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(BLACK_BRUSH);
    WndClass.lpszMenuName=NULL;
    WndClass.lpszClassName=lpszClass;
    WndClass.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
    RegisterClassEx (&WndClass);

    hWnd = CreateWindow ( lpszClass, lpszWindowName, WS_OVERLAPPEDWINDOW,
                                  0, 0, 800, 600, NULL ,(HMENU)NULL, hInstance, NULL);
    ShowWindow (hWnd,nCmdShow);
    UpdateWindow (hWnd);

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

```
LRESULT CALLBACK WndProc (HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC  hDC;
    TCHAR temp[] = TEXT("Hello world!");
    int x = 0, y = 0;

    switch(iMessage) {
    case WM_PAINT:
        hDC = BeginPaint (hWnd, &ps);
        TextOut (hDC, x, y, temp, lstrlen(temp));
        EndPaint (hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage (0);
        return 0;
    }
    return (DefWindowProc (hWnd, iMessage, wParam, lParam));
}
```



1. WinMain() 함수

- **WinMain() 함수 처리 내용**
  - 윈도우 클래스 만들기
  - 윈도우 클래스 구조체에 속성을 설정하고 등록하기 : 윈도우 함수, 아이콘, 커서, 배경색
  - 윈도우 만들기: 윈도우 좌표, 스타일
  - 윈도우를 화면에 띄우기
  - 윈도우에서 발생한 이벤트에 관한 메시지 보내기
    - 메시지 큐에 저장된 메시지를 꺼내 윈도우 프로시저로 보내 처리

WinMain()의 형식

- **WinMain 함수**

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
    HWND hWnd;
    MSG Message;
    WNDCLASSEX WndClass;
    g_hInst = hInstance;
```

  - WINAPI: 윈도우 프로그램이라는 의미
  - hInstance: 현재 실행중인 어플리케이션의 인스턴스 핸들
  - hPrevInstance: 동일한 어플리케이션이 실행중일 경우 이전에 실행된 프로그램의 인스턴스 핸들. Win32 어플리케이션의 경우 항상 NULL
  - lpszCmdLine: 커멘드라인 상에서 프로그램 구동 시 전달된 문자열
  - nCmdShow: 윈도우가 화면에 출력될 형태

- **인스턴스 (Instance)**
  - 어떤 대상이 메모리에 생성된 클래스의 실체

윈도우 클래스

- **윈도우클래스 생성하기**
  - 윈도우 클래스: 생성하는 윈도우의 형태를 정의하기 위해 사용하는 구조체

```
typedef  struct  _WndClassEX {
        UINT            cbSize;          //--- 본 구조체의 크기
        UINT            style;           //--- 출력 스타일
        WNDPROC         lpfnWndProc;     //--- 프로시저 함수
        int             cbClsExtra;      //--- 클래스 여분 메모리--- 사용안함
        int             cbWndExtra;      //--- 윈도우 여분 메모리--- 사용안함
        HANDLE          hInstance;       //--- 윈도우 인스턴스
        HICON           hIcon;           //--- 아이콘
        HCURSOR         hCursor;         //--- 커서
        HBRUSH          hbrBackground;   //--- 배경색
        LPCTSTR         lpszMenuName;    //--- 메뉴이름
        LPCTSTR         lpszClassName;   //--- 클래스 이름
        HICON           hIconSm;         //--- 작은 아이콘
} WndClassEX;
```

윈도우 클래스

속성 이름	의미	사용 가능 속성
cbSize	본 구조체의 크기	
style	윈도우 클래스의 스타일을 나타낸다. **Bitwise OR (|) 연산자를 이용하여 여러 개의 스타일을 OR**로 설정할 수 있다.	CS_HREDRAW / CS_VREDRAW: 작업 영역의 폭/높이가 변경되면 윈도우를 다시 그린다. CS_DBCLKS: 마우스 더블 클릭 메시지를 보낸다 CS_CLASSDC: 이 클래스로부터 만들어진 모든 윈도우가 하나의 DC를 공유한다. CS_OWNDC: 각 윈도우가 하나의 DC를 독점적으로 사용한다. CS_PARENTDC: 자식 윈도우가 부모 윈도우의 DC를
lpfnWndProc	윈도우의 메시지를 처리하는 윈도우 프로시저 함수 이름	
hInstance	이 윈도우 클래스를 사용하는 프로그램의 인스턴스 값	WinMain 함수의 인수로 전달된 hInstance 값을 사용
hIcon	실행 파일에 쓰일 아이콘 지정	IDI_APPLICATION / IDI_ASTERISK / IDI_EXCLAMATION / IDI_HAND / IDI_QUESTION/ IDI_ERROR / IDI_WARNING / IDI_INFORMATION
hCursor	윈도우에 쓰일 커서 지정	IDC_APPSTARTING / IDC_ARROW / IDC_CROSS / IDC_HAND / IDC_HELP/ IDD_IBEAM / IDC_SIZEALL / IDC_SIZENESW / IDC_SIZENS / IDC_SIZENWSE / IDC_SIZEWE / IDC_UPARROW / IDC_WAIT
hbrBackground	윈도우의 배경색, 기본색 또는 임의의 색을 설정할 수 있다.	GetStockObject 함수의 인자값에 BLACK_BRUSH / WHITE_BRUSH / DKGRAY_BRUSH / LTGRAY_BRUSH / GRAY_BRUSH
lpszMenuname	**메뉴의 이름, 컨트롤의 id로도 사용된다.**	
lpszClassName	**이 윈도우 클래스의 이름**	
hIconSm	작은 아이콘에 쓰일 아이콘을 지정	

윈도우 클래스 정의

- **윈도우 클래스 정의하기**

```
WndClassEX  WndClass ;                                          //--- 구조체 정의

LPCTSTR lpszClass = L"Window Class Name";
LPCTSTR lpszWindowName = L"Window Programming Lab";

WndClass.cbSize  = sizeof(WndClass) ;                           //--- 구조체 크기
WndClass.style = CS_HREDRAW | CS_VREDRAW ;                      //--- 윈도우 출력 스타일
WndClass.lpfnWndProc = WndProc ;                                //--- 프로시저 함수 명
WndClass.cbClsExtra = 0 ;                                       //--- O/S 사용 여분 메모리(Class)
WndClass.cbWndExtra = 0 ;                                       //--- O/S 사용 여분 메모리(Window)
WndClass.hInstance = hInstance ;                                //--- 응용 프로그램 ID
WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);               //--- 아이콘유형
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);                 //--- 커서 유형
WndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);   //--- 배경
WndClass.lpszMenuName = NULL ;                                  //--- 메뉴 이름
WndClass.lpszClassName = lpszClass;                             //--- 클래스 이름
WndClass.hIconSm = LoadIcon(NULL,IDI_APPLICATION);              //--- 작은 아이콘
```

윈도우 클래스 등록

- **윈도우 클래스를 운영체제에 등록한다.**

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
   HWND hWnd;
   MSG Message;
   WNDCLASSEX WndClass;
   g_hInst = hInstance;

   WndClass.cbSize = sizeof(WndClass);
   WndClass.style = CS_HREDRAW | CS_VREDRAW;
   WndClass.lpfnWndProc = (WNDPROC)WndProc;
   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(BLACK_BRUSH);
   WndClass.lpszMenuName = NULL;
   WndClass.lpszClassName = lpszClass;
   WndClass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
   RegisterClassEx(&WndClass);
```

- ATOM RegisterClassEx (CONST WndClassEX *lpwcx);
  - &lpwcx : 앞서 정의한 윈도우 클래스의 주소

윈도우 만들기

- **윈도우 만들기 함수**

```
HWND CreateWindow (                     //--- 윈도우 핸들 값 반환
        LPCTSTR lpClassName,            //--- 윈도우 클래스 이름
        LPCTSTR lpWindowName,           //--- 윈도우 타이틀 이름
        DWORD dwStyle,                  //--- 윈도우 스타일
        int x,                          //--- 윈도우 위치 x 좌표
        int y,                          //--- 윈도우 위치 y 좌표
        int nWidth,                     //--- 윈도우 가로 크기
        int nHeight,                    //--- 윈도우 세로 크기
        HWND hWndParent,                //--- 부모 윈도우 핸들
        HMENU hMenu,                    //--- 메뉴 핸들
        HINSTANCE hInstance,            //--- 응용 프로그램 인스턴스
        LPVOID lpParam                  //--- 생성 윈도우 정보
);
```

윈도우 만들기

속성 이름	의미	사용 가능 속성
lpClassName	윈도우 클래스에서 설정한 윈도우 클래스 이름	
lpWindowName	윈도우의 타이틀 이름	
dwStyle	윈도우의 다양한 스타일	WS_OVERLAPPED: 디폴트 윈도우 WS_CAPTION: 타이틀 바를 가진 윈도우 WS_HSCROLL / WS_VSCROLL: 수평/수직 스크롤 바 WS_MAXIMIZEBOX / WS_MINIMIZEBOX: 최대화/최소화 버튼 WS_SYSMENU: 시스템 메뉴 WS_THICKFRAME: 크기 조정이 가능한 두꺼운 경계선 WS_BORDER: 단선으로 된 경계선, 크기 조정 불가능 WS_POPUP: 팝업 윈도우 (WS_CHILD와 같이 쓸 수 없다) WS_CHILD: 차일드 윈도우 WS_VISIBLE: 윈도우를 만들자마자 화면에 출력 **WS_OVERLAPPEDWINDOW**: 가장 일반적인 윈도우 스타일 **WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX** WS_POPUPWINDOW: 일반적인 팝업 윈도우 WS_POPUP | WS_BORDER | WS_SYSMENU
x, y	윈도우의 좌표값 (좌측 상단 기준)	
nWidth / nHeight	윈도우의 가로, 세로 크기 (픽셀 단위)	
hWndParent	부모 윈도우 핸들, 부모가 없을 때는 NULL	
hMenu	윈도우의 상단에 붙는 메뉴의 핸들: 메뉴가 없을 때는 NULL	
hInstance	WinMain에서 받은 인스턴스 핸들	

윈도우 출력

- **윈도우의 출력 상태를 설정**
  - BOOL ShowWindow (HWND hWnd, int nCmdShow);
    - 윈도우의 보이기 상태를 지정한다.
    - hWnd: 윈도우 핸들
    - nCmdShow:
      - SW_HIDE : 윈도우를 숨기고 다른 윈도우를 활성상태로 만든다.
      - SW_MAXMIZE: 윈도우를 최대화
      - SW_MINIMIZE: 윈도우를 최소화하고 다른 윈도우를 활성 상태로
      - SW_SHOW: 윈도우를 나타내고 활성상태로 만든다.
      - SW_RESTORE: 최대/최소화를 원래 상태로 복원
  - BOOL UpdateWindow (HWND hWnd);
    - 윈도우 프로시저로 WM_PAINT 메시지를 보내 작업영역을 강제로 그리도록 한다. WM_PAINT 메시지를 곧바로 전달하므로 메시지 대기 순서에 상관없이 즉시 작업영역을 다시 그리도록 한다.

윈도우 만들고 출력하기

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
    HWND hWnd;
    MSG Message;
    WNDCLASSEX WndClass;
    g_hInst = hInstance;

    WndClass.cbSize = sizeof(WndClass);
    WndClass.style = CS_HREDRAW | CS_VREDRAW;
    WndClass.lpfnWndProc = (WNDPROC)WndProc;
    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;   // MAKEINTRESOURCE(IDR_MENU1);
    WndClass.lpszClassName = lpszClass;
    WndClass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
    RegisterClassEx(&WndClass);

    hWnd = CreateWindow(lpszClass, lpszWindowName, WS_OVERLAPPEDWINDOW|WS_HSCROLL|WS_VSCROLL, 0, 0, 800, 600, NULL, (HMENU)NULL, hInstance, NULL);
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

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

메시지 루프: 이벤트 메시지 보내기

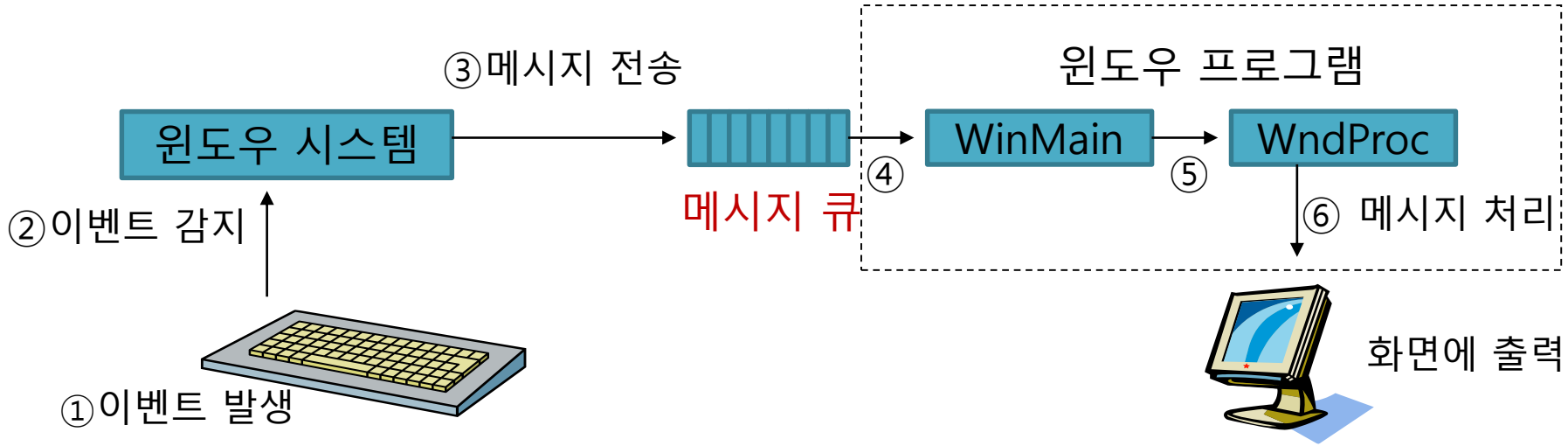
- **이벤트의 메시지를 실행하도록 메시지 처리 루프를 통해 메시지를 윈도우 프로시저에 보낸다.**

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

  - 메시지 처리 루프를 만들기 위한 함수들
    - BOOL GetMessage (LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax);
      - 메시지 큐로부터 메시지를 얻어오는 역할
      - 종료 메시지인 WM_QUIT 메시지가 들어올 때까지 계속 메시지를 얻어온다.
    - BOOL TranslateMessage (CONST MSG *lpMsg);
      - 키보드 입력 이벤트 중 문자 입력을 처리하는 함수로 단축키 명령어를 기본적인 이벤트로 번역 또는 변환
      - 예) Shift 'a'→ 대문자 'A'
    - LONG DispatchMessage (CONST MSG *lpmsg);
      - 실제적인 이벤트 처리 역할
      - GetMessage 함수로부터 전달된 메시지를 윈도우 프로시저로 보낸다.
      - WinMain → WinProc

이벤트 메시지 보내기

- 메시지 처리 과정



2. 윈도우 프로시저(Window Procedure) 함수

- **윈도우 프로시저: WinMain()에서 전달된 메시지를 처리하는 함수**
  - 메시지들을 처리하는 함수
  - 함수 프로토타입:

    LRESULT CALLBACK WndProc (HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);

    - hWnd: 메시지를 보내는 윈도우의 핸들
    - msg: 처리될 윈도우 메시지의 코드나 ID
    - wParam: 메시지 부가정보 (숫자, ID, 분류 등)
    - lParam: 메시지 부가정보 (숫자, ID, 분류 등)

    - LRESULT: win32환경에서 메시지 처리를 마친 후 OS에 신호를 주기 위한 값, long 타입
    - WPARAM: 핸들이나 정수값을 위해 사용하는 타입
    - LPARAM: 포인터 전달에 사용되는 타입

  - 윈도우로 전달되는 메시지를 처리하는 메시지 처리 함수로 사용자 정의 함수
  - 함수명이 꼭 WndProc일 필요는 없다.
  - 윈도우 속성설정 부분에서 wc.lpfnWndProc에 윈도우 프로시저 이름을 설정
  - 윈도우 프로시저는 항상 **DefWindowProc** 함수를 호출하며 마무리해야 한다.

    LRESULT DefWindowProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) 함수

    - WndProc 에서 처리하지 않은 나머지 메시지를 처리하도록 한다.
    - 예를 들어, 시스템 메뉴 메시지 처리 등

    - 콜백 함수(Callback): 이벤트가 발생했을 때 윈도우에 의해서 호출되는 것
      - 일반 함수 호출은 응용 프로그램이 운영체제에 내장된 함수를 호출하여 원하는 작업을 하는데, 콜백 함수는 거꾸로 운영 체제가 응용 프로그램을 부른다. (예, 타이머 함수, WndProc 함수)

윈도우 프로시저 함수

- 함수 형태

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hDC;

    switch (iMessage) {
    case WM_PAINT:
        hDC = BeginPaint(hWnd, &ps);
        EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;
    }
    return (DefWindowProc(hWnd, iMessage, wParam, lParam));
}
```

1과 2를 합쳐서, 윈도우 프로그램 작성하기

- **Include 파일 사용**
  - 윈도우 응용 프로그램에 필요한 파일을 포함시킨다.
  - windows.h / windowsx.h
- **메인 함수: 등록 부분 → WinMain ()**
  - 윈도우즈 클래스 구조체에 값을 지정
  - 윈도우즈 클래스 등록
  - 윈도우즈 생성
  - 윈도우 출력
  - 이벤트 루프 처리하기
- **메시지 처리 함수: 메시지 처리 부분 → WndProc ()**
  - 사용자와 시스템이 보내오는 메시지를 처리한다.
  - DefWindowProc 함수를 호출하며 마무리 한다.

- **윈도우 프로그램에서는 WinMain과 WndProc이 모두 있어야 한다.**

윈도우 프로그램 작성하기

```
#include <windows.h>                    //--- 윈도우 헤더 파일
#include <tchar.h>

HINSTANCE g_hInst;
LPCTSTR lpszClass = L"Window Class Name";
LPCTSTR lpszWindowName = L"Window Programming Lab";

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
	HWND hWnd;
	MSG Message;
	WNDCLASSEX WndClass;
	g_hInst = hInstance;

	WndClass.cbSize = sizeof(WndClass);
	WndClass.style = CS_HREDRAW | CS_VREDRAW;
	WndClass.lpfnWndProc = (WNDPROC)WndProc;
	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(BLACK_BRUSH);
	WndClass.lpszMenuName = NULL;
	WndClass.lpszClassName = lpszClass;
	WndClass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
	RegisterClassEx(&WndClass);

	hWnd = CreateWindow (lpszClass, lpszWindowName, WS_OVERLAPPEDWINDOW, 0, 0, 800, 600, NULL, (HMENU)NULL, hInstance, NULL);
	ShowWindow (hWnd, nCmdShow);
	UpdateWindow (hWnd);

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

윈도우 프로그램 작성하기

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
        PAINTSTRUCT ps;
        HDC  hDC;

        //--- 메시지 처리하기
        switch (uMsg) {
             case WM_CREATE:
                break;
             case WM_PAINT:
                hDC = BeginPaint(hWnd, &ps);
                EndPaint(hWnd, &ps);
                break;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
        }
        return DefWindowProc (hWnd, uMsg, wParam, lParam);          //--- 위의 세 메시지 외의 나머지 메시지는 OS로
}
```

변수 명명법

- **변수 명명법**
  - 변수 이름을 이해하기 쉽게 하기 위해 대, 소문자를 혼합하여 길게 사용
  - 변수명을 만들 때, 변수명 앞에 데이터형 접두어를 붙여 어떤 데이터 타입을 갖는지 알 수 있다.

접두어	의미	
cb	Count of Bytes	바이트 수
dw	Double word	부호 없는 long 정수
h	Handle	핸들
sz	Null terminated	널 종료 문자열
ch	Character	문자
a	Array	배열
w	Word	부호 없는 정수형
i	Integer	정수형
p, lp	Long pointer	포인터형
b	bool	논리형
l	Long integer	Long 정수형

데이터 타입

- **윈도우 프로그램에서 정의된 데이터 타입**

데이터 타입	의미	데이터 타입	의미
BOOL	TRUE 또는 FALSE	APIENTRY	시스템 함수를 호출하는 호출 규약.
BYTE	8 bit unsigned int	WINAPI	시스템 함수를 호출하는 호출 규약
DWORD	32 bit unsigned int	HBITMAP	비트맵 핸들
WORD	16 bit unsigned int	HBRUSH	브러시 핸들
UINT	unsigned int	HDC	DC 핸들
LPARAM	32 bit 메시지 파라미터	HGDIOBJ	GDI Object 핸들
WPARAM	16 bit 메시지 파라미터	HINSTANCE	인스턴스 핸들
HANDLE	오브젝트 핸들	COLORREF	Red, green, blue 컬러값을 가진 32비트
LPVOID	어떤 타입이든 가리키는 void 포인터	CALLBACK	콜백 함수를 위한 호출 규약

데이터 타입 중 핸들 (Handle)

- **핸들(Handle)**
  - 프로그램에서 현재 사용중인 객체 (윈도우, 커서, 아이콘, 메뉴 등)들을 구분하기 위해 윈도우 OS가 부여하는 고유
  - 32비트 정수형
  - 핸들값은 접두어 h로 시작한다.
  - 핸들은 운영체제가 발급하며 사용자는 사용만 한다.
  - 같은 종류의 핸들끼리는 절대 중복된 값을 가지지 않는다.
  - 핸들은 단순한 구분자이므로 핸들에 어떤 값이 들어가 있는지 알 필요가 없다!

핸들 자료형	의미
HINSTANCE	인스턴스 핸들
HWND	윈도우 핸들
HDC	장치 컨텍스트 핸들
HBRUSH	브러시 핸들
HPEN	펜 핸들
HFONT	폰트 핸들
HBITMAP	비트맵 핸들

데이터 타입 중 문자형

- **윈도우 프로그램에서 문자형 데이터 타입**
  - LP: Long Pointer (32bit pointer)
  - C: Constant
  - STR: string
  - W: wide_char

데이터 타입	의미	
LPSTR	널 문자로 끝나는 윈도우 문자열 포인터 (Long Pointer String)	char *
LPCSTR	널 문자로 끝나는 상수형 문자열 포인터	const char *
LPWSTR	유니코드 형의 널 문자로 끝나는 윈도우 문자열 포인터	w_char *
LPCWSTR	유니코드 형의 널 문자로 끝나는 상수형 문자열 포인터	const w_char *
LPTSTR	일반 및 유니코드 형의 문자열 포인터	TCHAR *
LPCTSTR	일반 및 유니코드 형의 상수형 문자열 포인터	const TCHAR *

유니코드

- **문자 타입**
  - 멀티 바이트: 한 글자를 저장하기 위해 2바이트 이상 사용할 수 있게 해 줌
  - 유니 코드: 2 바이트로 문자를 저장
    - 문자열의 상수 타입 앞에 L 을 붙인다.
    - char 대신 wchar 자료형으로 사용
  - TCHAR라는 새로운 타입으로 사용: _TEXT ("...") 함수를 사용하여 타입 변환
    - 프로젝트 속성에서 문자 집합을 "유니코드 문자 집합 사용"으로 설정
- **사용 하기**
  - 필요한 헤더파일 추가
    - #include <TCHAR.H>
  - 문자열을 선언: TCHAR 타입
- **문자열 변환 함수**
  - TEXT 함수, L, _T 매크로 함수 사용
    - TCHAR temp[] =TEXT ("Hello world1");
    - TCHAR temp2[] = L "Hello world2";
    - TCHAR temp3[] = _T ("Hello world3");

1장에서 기억해야 할 내용들

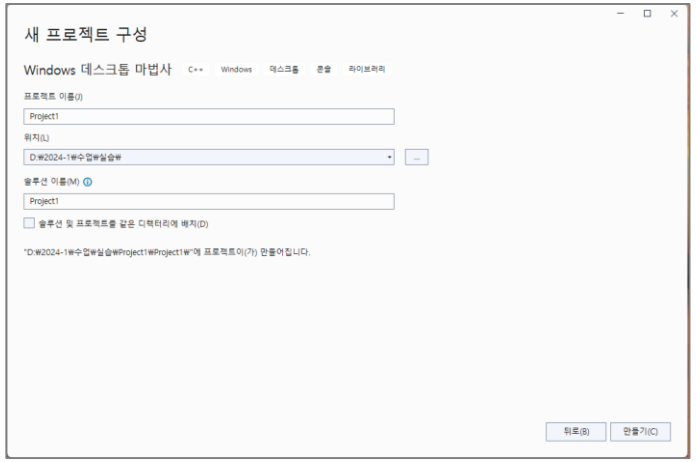
- **윈도우 프로그램의 특징**
  - GUI
  - 메시지 기반으로 구동 됨
  - 장치 독립적
  - 리소스가 분리되어 있다
  - 멀티 태스킹
- **윈도우 프로그래밍 개념**
  - API
  - 메시지 (이벤트)
  - 윈도우 프로시저
- **윈도우 프로그래밍**
  - 윈도우 프로그램 구성
  - 윈도우 프로그램이 어떻게 실행되는지 이해!

윈도우 프로젝트 만들기

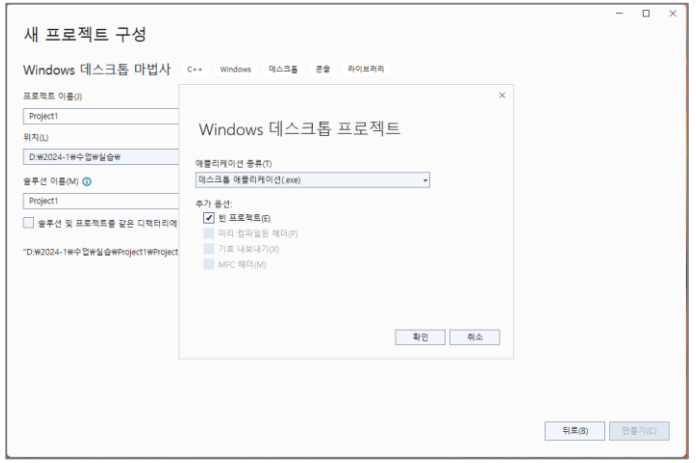
- 윈도우 기반의 프로젝트 만들기 (VS2022 기준)

"새 프로젝트 만들기" 선택

"Windows 데스크톱 마법사" 선택



프로젝트 이름과 위치 설정 후 만들기

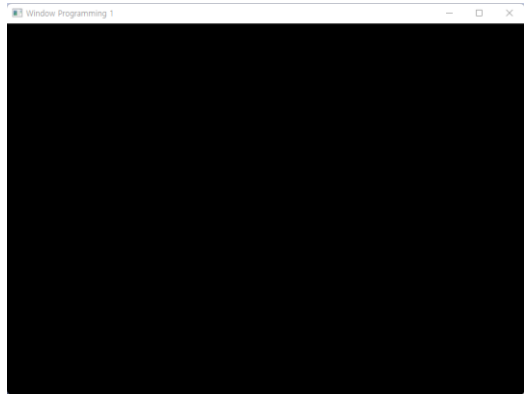


애플리케이션 종류: "데스크톱 애플리케이션" 선택
추가 옵션: 빈 프로젝트 선택

실습 1-1

- **제목**
  - 간단한 윈도우 만들기

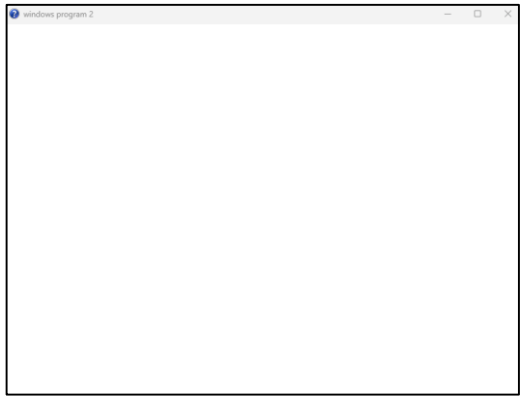
- **내용**
  - 윈도우의 타이틀을 “windows program 1”으로 설정하고,
  - 기본 윈도우 형태로
  - 배경색을 검정색으로
  - 윈도우의 위치를 (0, 0)에, 윈도우의 크기를 1280*800 으로 만들어 출력하기.



실습 1-2

- **제목**
  - 간단한 윈도우 만들기 2

- **내용**
  - 윈도우의 타이틀을 “windows program 2”로 설정하고,
  - 시스템 메뉴, 최대, 최소화 버튼, 수평/수직 스크롤 바를 가지고 있고, 크기가 조정 가능한
  - 배경색을 하얀색으로
  - 윈도우를 위치 (100, 50)에, 윈도우의 크기를 800*600 으로 만들어 출력하기.
  - 스몰 아이콘을 물음표로, 커서는 손 모양으로 설정하기



실습 1-3

- **제목**
  - 윈도우 프로그래밍 시간에 만들어 보면 좋을 것 같은 게임은 어떤 것이 있을까

- **내용**
  - 윈도우 프로그래밍 수업시간에 실습 또는 숙제로 만들어 보면 좋을 것 같은 2차원 게임 리스트 작성하기
  - 최소 게임 2개의 이름을 제시하기
  - 작성 내용
    - 게임 제목
    - 게임을 소개하고 선택한 이유를 간단히 적기
    - 게임의 내용을 볼 수 있는 유튜브 주소나 레퍼런스 사이트 주소

    (** 여러분들이 실습 1-3에서 제시하는 게임은 이번 학기가 아니라 다음 학기 수업에 반영될 수 있음)

- **제출:**
  - 날짜: 3월 22일 (금)까지
  - 제출 방법: 이클래스 과제 항목에 문서 제출. 문서 파일로 작성하기 (한글 또는 워드 파일)

- 실습 1-1, 1-2는 1점
- 실습 1-3은 2점으로 채점