1장 윈도우 프로그래밍 기초

2018년도 1학기

우리가 그 동안 해왔던 많은 게임들

2차원 또는 3차원 게임들

대부분
윈도우 기반의
게임들

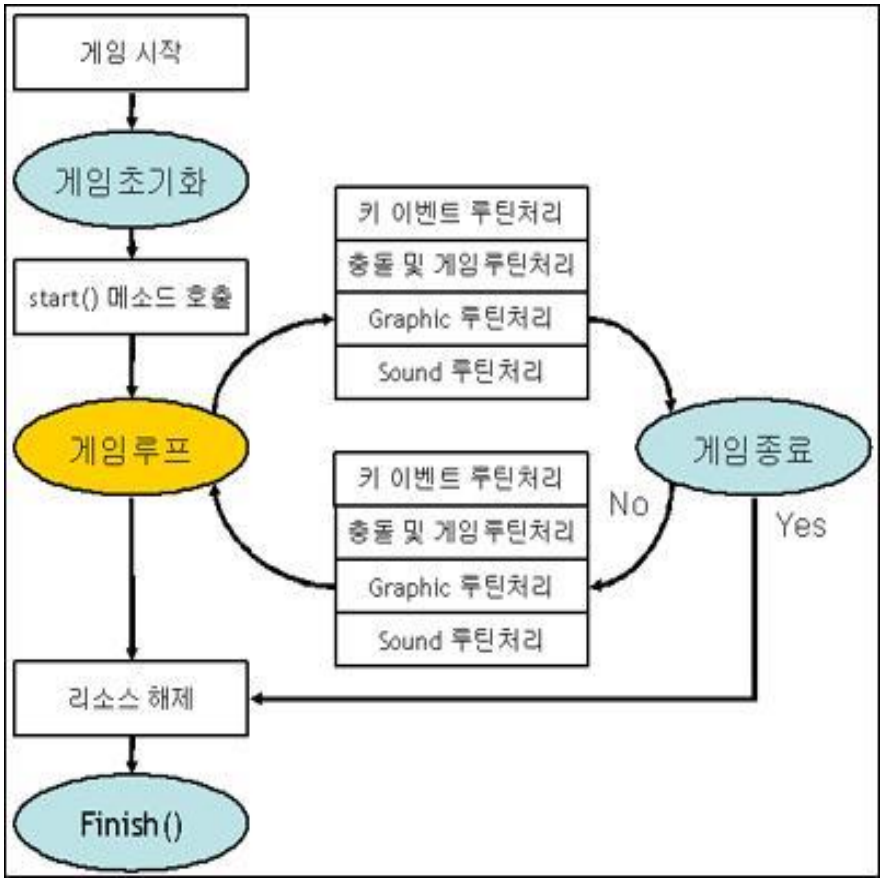
키보드나 마우스
이용해 게임 진행

스프라이트나 모델링 데이
터를 이용한 애니메이션

버튼이나 스크롤 등의 컨
트롤사용

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 기록하기

우리가 이번 학기 배울 내용들

- **윈도우 띄워 데이터 출력하기**
- **키보드와 마우스 입력 받기**
- **비트맵 이미지 사용하기**
- **타이머를 이용한 애니메이션 만들기**

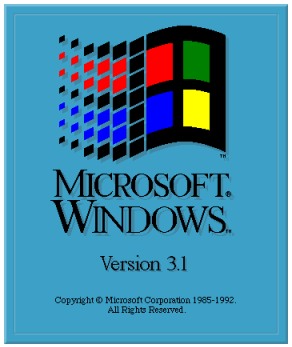
- **다양한 컨트롤 사용하기**
  - 버튼이나 선택 컨트롤 등을 사용
- **파일에 데이터 저장하기**
  - 데이터 기록하기
- **다중 윈도우 띄우기**

- **Map Tool 만들기**
- **2차원 게임 만들기**

윈도우의 역사

- **MS-DOS(Disk Operating System) 시대**
  - Microsoft사에서 개발
  - 1981년 IBM이 16비트 운영체제인 MS-DOS를 채택하면서 DOS 시스템이 사용됨

- **Windows 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

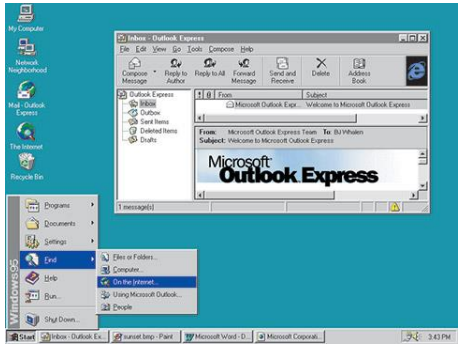


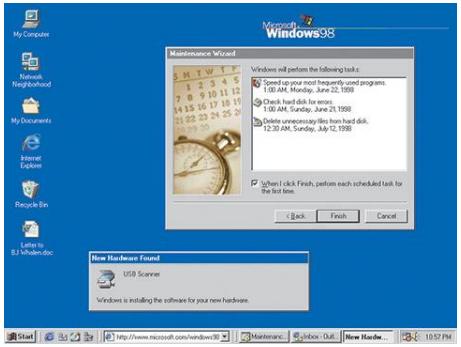
윈도우의 역사

- **Windows 95**
  - 1995년 8월 windows95 발표
  - 기존의 윈도우 버전까지는 MS-DOS 기반
  - Windows 95 이후부터는 현재 사용되는 윈도우 인터페이스의 기반

- **Windows 98**
  - Windows 95 기반에 인터넷, 멀티미디어, 파일시스템 등의 기능을 개선한 버전

- **Windows XP, Windows vista, Windows 7, Windows 8, Windows 10**

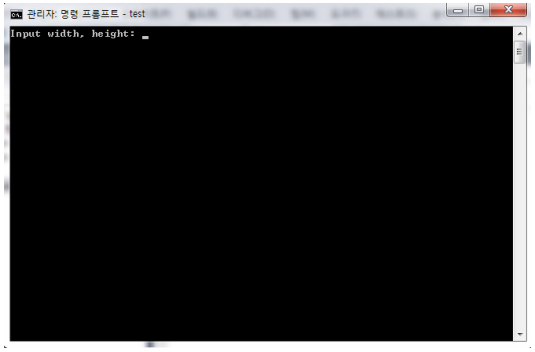


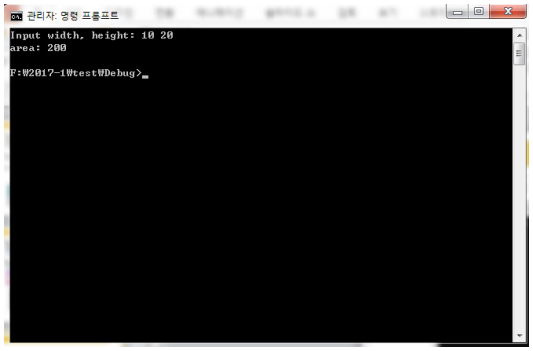


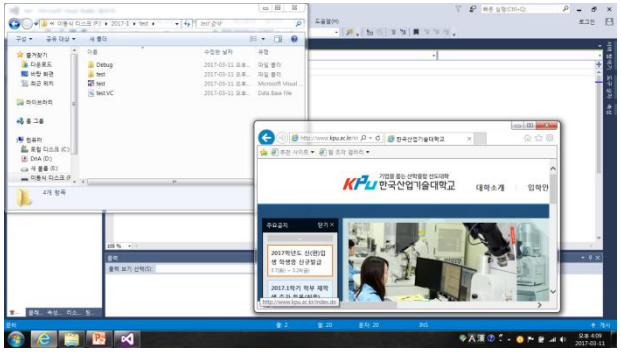
운영 체제: DOS, Window

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







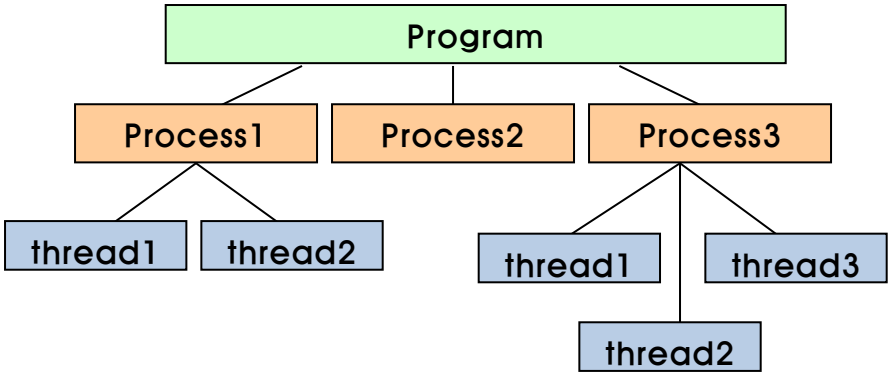
윈도우 운영 체제의 특징

- **윈도우 운영 체제의 특징**
  - 그래픽 사용자 인터페이스 (GUI) 기반의 운영체제
    - 픽셀 단위의 그래픽 기반 운영 체제
    - 아이콘 , 탐색기, 메뉴, 스크롤 등을 마우스를 이용하여 사용
  - 메시지 구동 (이벤트 구동) 시스템
    - 메시지: 윈도우가 어플리케이션에게 보내는 알림
    - 운영 체제로부터 메시지를 받아 동작한다.
      - 외부에서 발생한 일들을 윈도우 OS가 감지하여 해당 프로그램에 메시지를 전달한다.
  - 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 없음
  - 리소스 분리
    - 코드 이외의 데이터가 분리되어 있다.
  - 멀티태스킹 (Multi-tasking)
    - DOS는 single-tasking: 한번에 하나의 프로그램만 실행
    - 동시에 여러 개의 프로그램을 실행시킬 수 있다.
      - 워드 프로세서로 문서를 작성하면서 스프레드 시트로 데이터를 참조할 수 있다.
    - CPU는 하나이기 때문에, 실행할 기회를 여러 개의 프로세스에게 순서대로 나누어 주어서 동시에 여러 개가 함께 실행되는 것처럼 보인다.

윈도우 운영 체제의 특징

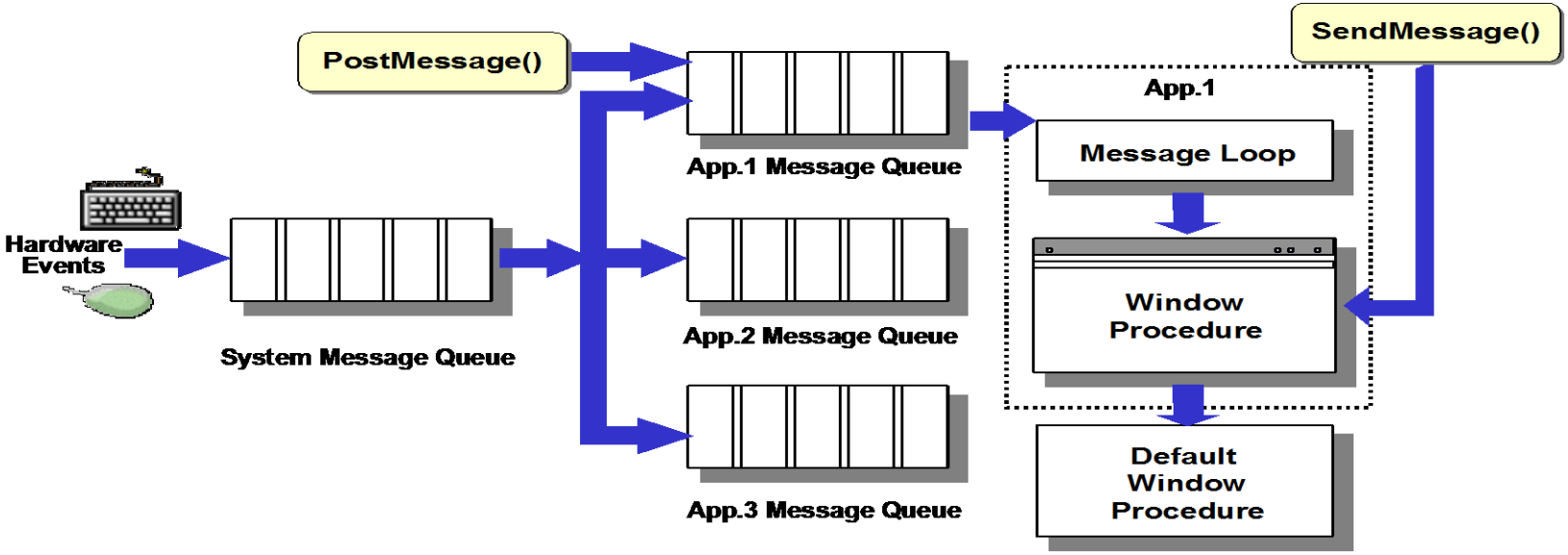
- **프로그램(program), 프로세스(process)**
  - 프로그램(program): 파일이 디스크에 저장된 내용과 같은 수동적인 실체 (passive entity), 실제 실행되는 파일
  - 프로세스(process): 프로그램에 의한 작업의 기본 단위, 즉 프로그램에서 지시하는 명령을 읽어 들여 실행하고, 종료 지시를 만나면 처리를 종료하는 등의 일을 실행되는 하나의 작업 단위, 하나의 프로그램을 여러 번 구동하면 여러 개의 프로세스로 메모리에 적재되어 실행됨

- **멀티 쓰레딩 (Multi-threading)**
  - 쓰레드(thread): 프로세스를 구성하는 프로세스보다 작은 태스크(task) 단위
  - 프로세스 하나가 쓰레드 여러 개를 병행 처리할 수 있는데 이것을 **멀티 쓰레드**라고 한다.
    - 앞에서는 사용자의 조작에 응답하고 백그라운드에서 인쇄 등의 시간이 걸리는 작업을 처리하는 기능
  - 각각의 쓰레드는 자신만의 코드를 수행한다.
    - 쓰레드의 컨텍스트 정보가 적기 때문에 빠르게 만들 수 있다.



윈도우에서 사용자 입력 구조

- **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 메시지를 사용하여 통지**한다.
    - 윈도우 애플리케이션은 자신이 소유한 윈도우에 대해 **윈도우 프로시저 함수** 준비
    - 윈도우 프로시저는 윈도우에서 다양한 이벤트가 발생했을 때 이를 처리한다.
    - **윈도우에서는 이벤트가 발생했을 때마다 메시지 내용을 인자로 넘겨서 윈도우 프로시저를 호출한다.**



윈도우에서 사용자 입력 구조

- **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 전달인자 1
    LPARAM lParam;          // 메시지 전달인자 2
    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 쓰레드 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.**
  - 각각의 API는 C언어로 기술된 응용 프로그램을 위한 함수들 모임
  - 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

  - MFC (Microsoft Foundation Class): 목적별로 복수의 API를 모아 프로그램 코드로 간접적으로 호출하기 위한 기능 제공

윈도우 프로그램 기본 개념들

- **API(Application Programming Interface)**
  - 응용 프로그램이 OS나 데이터베이스 관리 시스템 (DBMS)등과 통신할 때 사용되는 언어나 메시지 형식
  - 응용 프로그램 개발자를 위해 제공하는 함수 집합
  - 많은 종류의 API중 32비트 OS인 윈도우에서 제공하는 API를 Win32 API라고 한다.

- **SDK(Software Development Kit)**
  - 윈도우용 응용 프로그램 개발도구
  - 각종 편집 툴, 라이브러리, 헤더파일, 도움말, 예제 프로그램들로 구성되어 있다.
  - 그래픽 처리, 데이터베이스 접근, 동적링크 처리 등

윈도우 프로그램 기본 개념들

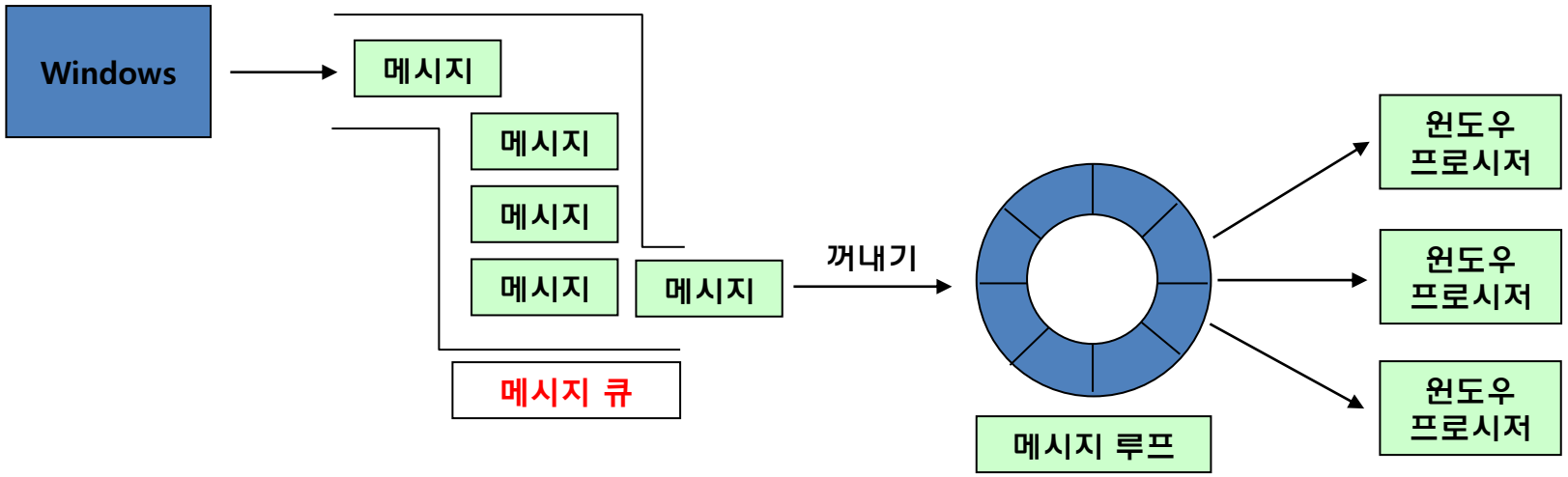
- **이벤트(Event)와 메시지(Message)**
  - *이벤트: 사용자가 키보드를 누르거나 마우스 버튼을 클릭할 때, 툴 바의 버튼을 누르거나 윈도우의 크기를 조절하는 등의 기계적인 조작에 의해 발생*
  - *이벤트가 발생하면 윈도우 OS는 이를 감지하여 해당 프로그램으로 메시지를 전달*
  - 마우스 누름(이벤트) -> WM_LBUTTONDOWN 메시지로 변환
    - 윈도우 메시지 유형들은 모두 "WM_"로 시작

윈도우 메시지 유형	이벤트 (발생하는 상황)
WM_CREATE	윈도우가 생성될 때
WM_ACTIVATE	윈도우가 활성화되거나 비 활성화 될 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_SIZE	윈도우의 크기가 변경되었을 때
WM_MOVE	윈도우가 이동되었을 때
WM_TIMER	설정된 타이머 시간이 되었을 때
WM_DESTROY	윈도우가 없어질 때

<여러 종류의 메시지들>

윈도우 프로그램 기본 개념들

- **메시지 큐(Message Queue)**
  - FIFO (First In First Out)
  - 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 기본 개념들

- **메시지 루프(Message Loop)**
  - 윈도우 OS가 프로그램에 전달한 메시지를 받아들여 분석하는 무한 루프
  - 일반적인 메시지 루프

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

  - GetMessage() 함수가 FALSE를 리턴 (프로그램을 종료하라는 WM_QUIT 메시지일 경우)할 때까지 메시지 큐로부터 메시지를 얻어와 처리

- **윈도우 프로시저(Window Procedure)**
  - 메시지 루프에서 해석한 메시지를 구체적으로 처리하는 기능을 하는 소스 부분
  - OS가 호출하는 콜백 함수(Callback function)
    - **콜백 함수: OS로부터 호출되는 함수 (윈도우 프로시저)**
  - 콜백 함수는 함수 앞에 키워드 CALLBACK을 쓰며 호출은 윈도우 OS가 함
  - 함수의 이름은 프로그래머가 마음대로 지정할 수 있다.
  - 함수 프로토타입
    - LRESULT **CALLBACK** WndPrc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

윈도우 프로그램

- 일반적인 윈도우즈 프로그램은 **C/C++언어 소스부분**과 **리소스 스크립트 파일(.rc), 관련 헤더**와 **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

- C/C++언어 소스부분은
  - WinMain()함수 한 개와 한 개 이상의 윈도우 프로시저 함수로 구성된다.

```
#include <windows.h>

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


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

메인 부분

- 윈도우를 만든다.
- 윈도우를 띄운다.
- 윈도우/응용 프로그램에서 발생하는 모든 메시지 전송
- **WinMain () 함수**

메시지 처리 부분

- 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

HINSTANCE g_hInst;
LPCTSTR lpszClass = "Window Class Name";

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, "window program",
                  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;
    char temp[] = "Hello world!";
    int x = 0, y = 0;

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

WinMain()의 처리내용

- 윈도우 클래스 만들기 : 윈도우 함수, 아이콘, 커서, 배경색
- 윈도우 클래스를 등록하기
- 윈도우 만들기 : 윈도우 좌표, 스타일
- 윈도우를 화면에 띄우기

- 윈도우에서 발생한 이벤트에 관한 메시지 보내기

WinMain()의 형식

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)

- 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
- szCmdLine: 커멘드라인 상에서 프로그램 구동 시 전달된 문자열
- iCmdShow: 윈도우가 화면에 출력될 형태

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

- **핸들(Handle)**
  - 프로그램에서 현재 사용중인 객체 (윈도우, 커서, 아이콘, 메뉴 등)들을 구분하기 위해 윈도우 OS가 부여하는 고유 번호
    - 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!

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;
```

- **구조체 크기:**
  - 본 구조체의 크기
- **프로시저 함수:**
  - 윈도우의 메시지를 처리하는 윈도우 프로시저 함수 이름
- **인스턴스 핸들:**
  - 이 윈도우 클래스를 사용하는 프로그램의 인스턴스 값
  - WinMain 함수의 인수로 전달된 hInstance 값을 사용
- **윈도우 클래스 이름:**
  - 이 윈도우 클래스의 이름

윈도우 클래스 정의

- **백그라운드:**
  - 윈도우의 배경색, 기본색 또는 임의의 색을 설정할 수 있다.
- **윈도우 출력 스타일**
  - 윈도우 클래스의 스타일을 나타낸다. Bitwise OR (|) 연산자를 이용하여 여러 개의 스타일을 OR로 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 사용한다.
- **아이콘:**
  - 실행 파일에 쓰일 아이콘 지정
    - IDI_APPLICATION/IDI_ASTERISK/IDI_EXCLAMATION/IDI_HAND/IDI_QUESTION
- **커서:**
  - 윈도우에 쓰일 커서 지정
    - IDC_APPSTARTING/IDC_ARROW/IDC_CROSS/IDC_HAND/IDC_HELP
- **스몰 아이콘:**
  - 윈도우 캡션에 쓰일 아이콘을 지정

윈도우 클래스 정의

```
WNDCLASSEX  wndclass ;                                                  // 구조체 정의
LPCTSTR lpszClass = "Window Class Name";

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.**

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
);
```

  - lpClassName : 윈도우 클래스에서 설정한 윈도우 클래스 이름
  - lpWindowName: 윈도우의 타이틀 이름
  - dwStyle: 윈도우의 다양한 스타일
  - 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 만들기

- dwStyle: 윈도우 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

윈도우 만들기

```
hwnd = CreateWindow {                   // 윈도우가 생성되면 윈도우의 핸들(hwnd)이 반환됨
            lpszClass,                  // 윈도우 클래스 이름
            "Window Title Name",        // 윈도우 타이틀 이름
            WS_OVERLAPPEDWINDOW,        // 윈도우 스타일
            0,                          // 윈도우 위치, x좌표              ---*
            0,                          // 윈도우 위치, y좌표              ---*
            800,                        // 윈도우 가로(폭) 크기            ---*
            600,                        // 윈도우 세로(높이) 크기          ---*
            NULL,                       // 부모 윈도우 핸들
            NULL,                       // 메뉴 핸들
            hInstance,                  // 응용 프로그램 인스턴스
            NULL) ;                     // 생성된 윈도우 정보

ShowWindow(hwnd, nCmdShow);             // 윈도우의 화면 출력
UpdateWindow(hwnd);                     // O/S에 WM_PAINT  메시지 전송
```

// *: 윈도우가 정하는 기본 값을 사용하려고 할 때는 CW_USEDEFAULT 상수를 사용할 수 있다.

윈도우 만들기

- **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.

이벤트 메시지 보내기

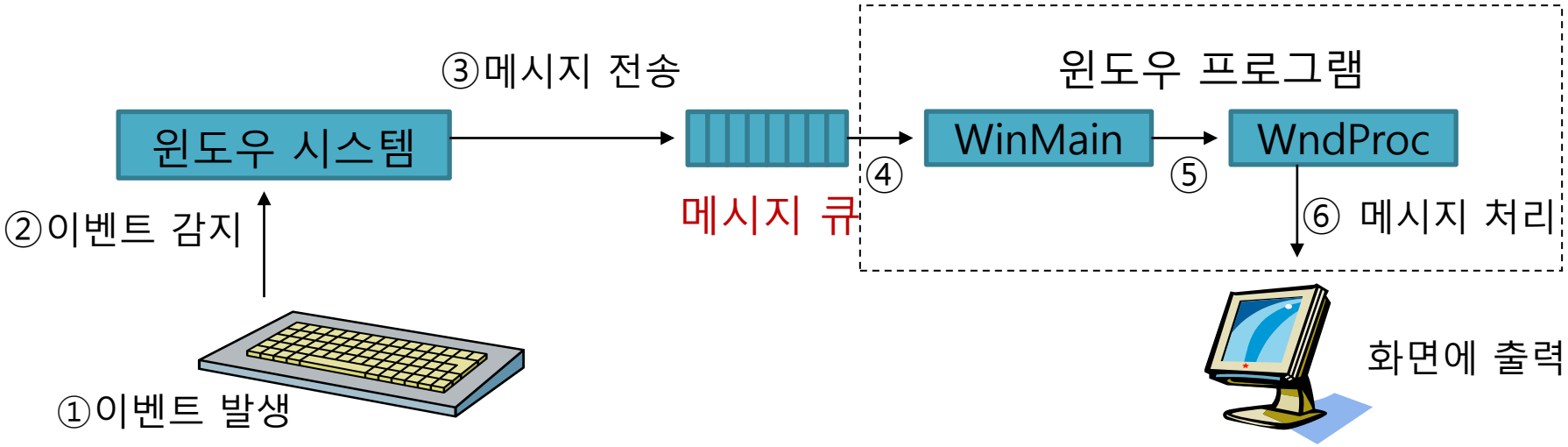
- **이벤트의 메시지를 실행하도록 메시지 처리 루프를 통해 메시지를 윈도우 프로시저에 보낸다.**

```
while (GetMessage (&msg, NULL, 0, 0))
 {                              // 윈도우 프로시저에서 PostQuitMessage() 호출 때까지 처리
    TranslateMessage (&msg) ;                  // Shift 'a'→ 대문자 'A'
    DispatchMessage (&msg) ;                   // WinMain → WinProc
 }
```

  - 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
    - LONG DispatchMessage (CONST MSG *lpmsg);
      - 실제적인 이벤트 처리 역할
      - GetMessage 함수로부터 전달된 메시지를 윈도우 프로시저로 보낸다.

이벤트 메시지 보내기

- **메시지 처리 과정**



윈도우 프로시저(Window Procedure)

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

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg,
        WPARAM wParam, LPARAM lParam)
{
     switch (iMsg) // 메시지 번호
     {
        case WM_CREATE:  // 메시지에 따라 처리
                        break;
        case WM_DESTROY:
                        PostQuitMessage (0) ;
                        break;
     } //처리할 메시지만 case문에 나열

     return DefWindowProc (hwnd, iMsg, wParam, lParam);
          // CASE에서 정의되지 않은 메시지는 커널이 처리하도록 메시지 전달
}
```

WndProc()

- **윈도우 프로시저 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

  - 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)

WndProc()

- **WndProc 함수**
  - 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

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.

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

윈도우 프로그램 작성하기

```
#include <windows.h>

HINSTANCE g_hInst;
LPCTSTR lpszClass = "Window Class Name";

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

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

    // 윈도우 클래스 구조체 값 설정
    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);

    // 윈도우 클래스 등록
    RegisterClassEX (&WndClass);

    // 윈도우 생성
    hWnd = CreateWindow ( lpszClass, "window API", WS_OVERLAPPEDWINDOW,  0, 0, 800, 600, NULL ,(HMENU)NULL, hInstance, NULL);
    // 윈도우 출력
    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);

    // 이벤트 루프 처리
    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)
{
        // 메시지 처리하기
        switch (uMsg) {
        case WM_CREATE:
                break;
        case WM_PAINT:
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
        break;
        }
        return DefWindowProc (hWnd, uMsg, wParam, lParam);          // 위의 세 메시지 외의 나머지 메시지는 OS로
}
```

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!

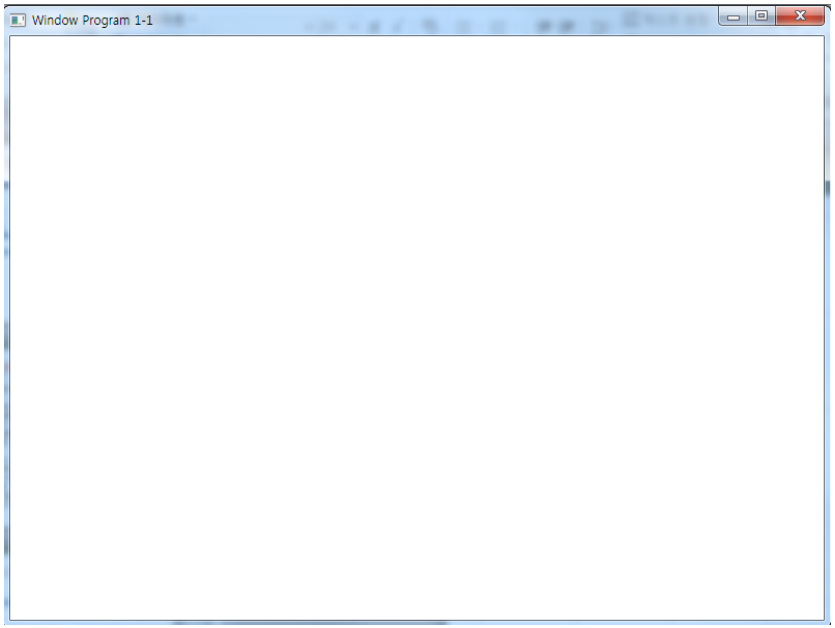
실습 1-1

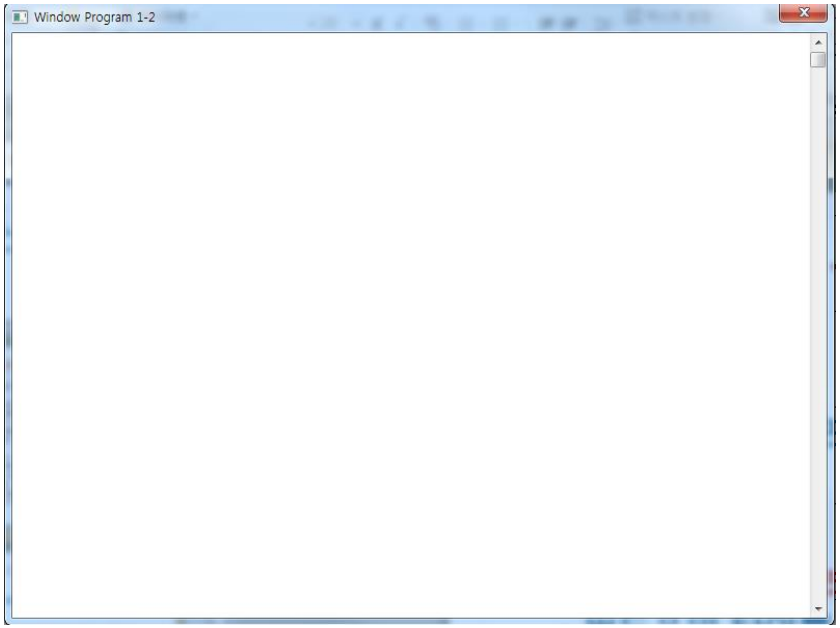
- **제목**
  - 간단한 윈도우 프로그램 2개를 작성하여 실행하기

- **내용**
  - 윈도우 1:
    - 윈도우의 타이틀을 "windows program 1-1"으로 설정하시오.
    - 시스템 메뉴를 가지고, 윈도우의 위치를 (0, 0)으로 설정하시오.
    - 윈도우의 크기를 1280*800 으로 설정하시오.

  - 윈도우 2:
    - 크기를 조절할 수 있고,
    - 윈도우의 타이틀을 "windows program 1-2"로 설정하고,
    - 시스템 메뉴, 최대, 최소화 버튼, 수평/수직 스크롤 바를 가지고 있고,
    - 윈도우를 위치 (100, 50)에 크기 800*600으로 만들어 띄우시오.

실습 1-1





실습 1-2

- **제목**
  - 게임 기획서 작성하기

- **내용**
  - 만들어보고 싶은 2D 게임의 기획서를 최소 2페이지 작성
  - 기획서 내용
    - 게임 제목
    - 게임의 내용 및 진행 방법
    - 게임의 특징 (왜 이런 게임을 만들고 싶었나)
    - 비슷한 게임의 스크린 샷 추가
  - 다음 시간까지 출력하여 제출