버전 1.0

2022.10.23



※현재 포토폴리오에서는 ‘코드를 작성한 이유’와 ‘설계’에 관해서 중점적으로 작성하였습니다.

작성자: 김경주

## WInAPI(윈도우 창)

### WinAPI 클래스를 main.cpp에서 분리한 이유:

→ 코드의 **재사용성**을 위해. 멀티 플랫폼이 대세이고, 혹시 WinAPI를 제외한 다른 플랫폼을 이용하고 싶을 수도 있기 때문에 코드를 분리하여서 사용

### 힘들었던 점:

졸업과제로 당시 directx9으로 엔진 만들 당시, 이런 구조의 코드를 작성하고 싶어서 mfc 코드를 뜯어서 적용시켜 봤으나 링크 에러로 고민을 많이 함.

### 에러 및 해결방법:

WindowProc 링크 에러  
→ win 프로시저 함수에 static을 적용하여서 this-> 라는 포인트를 제거함으로 해결  
 lnk2019 error  
→ (속성/링커/시스템) 하위 시스템을 콘솔에서 창으로 변경

#pragma once //미리컴파일된 해더

#ifndef STDAFX\_H // STDAFX\_H가 정의 안되어 있다면

#define STDAFX\_H // STDAFX\_H를 정의

#endif //가정 끝(pragma once)와 비슷한 효과 헤더파일이 중복되어 읽어도 괜찮도록 만들어 줌

#include <windows.h> //winAPI

#include <string> // string변수

stdafx.h(c++)

#pragma once

#include "DirectX12Base.h"

class WinAPI

{

public:

WinAPI(); //생성자

~WinAPI(); //소멸자

static bool Init( HINSTANCE hInstance, int nCmdShow); //초기화

static int Run(); //실행

static HWND GetHwnd(); //window 핸들 얻기

protected:

static LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam); //윈도우 프로시져

private:

static HWND WinAPI\_hwnd; //윈도우 핸들

};

WinAPI.h(c++)

#include "stdafx.h"

#include "WinAPI.h"

HWND WinAPI::WinAPI\_hwnd = nullptr; //핸들 값 초기화

WinAPI::WinAPI() //생성자

{

}

WinAPI::~WinAPI() //소멸자

}

bool WinAPI::Init( HINSTANCE hInstance, int nCmdShow) //초기값

{

//명령줄 매게변수 구분

int argc;

LPWSTR\* argv = CommandLineToArgvW(GetCommandLineW(), &argc);

LocalFree(argv);

// 윈도우 클라스 초기화

WNDCLASSEX windowClass = { 0 };

windowClass.cbSize = sizeof(WNDCLASSEX);

windowClass.style = CS\_HREDRAW | CS\_VREDRAW;

windowClass.lpfnWndProc = WindowProc;

windowClass.hInstance = hInstance;

windowClass.hCursor = LoadCursor(NULL, IDC\_ARROW);

windowClass.lpszClassName = L"DirectX12MiniEngine";

RegisterClassEx(&windowClass);

RECT windowRect = { 0, 0, 1200, 900 }; //윈도우 창범위

AdjustWindowRect(&windowRect, WS\_OVERLAPPEDWINDOW, FALSE);

// 창과 핸들 만듦

WinAPI\_hwnd = CreateWindow(

windowClass.lpszClassName,

L"DirectX12MiniEngine",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT,

CW\_USEDEFAULT,

windowRect.right - windowRect.left,

windowRect.bottom - windowRect.top,

nullptr, // 부모창 없음

nullptr, // 메뉴 사용하지 않음

hInstance,

nullptr);

ShowWindow(WinAPI\_hwnd, nCmdShow); //윈도우 보여주기

return true;

}

WinAPI.cpp(c++)

//WinAPI실행

int WinAPI::Run( )

{

// 메인 루프

MSG msg = { 0 };

while (msg.message != WM\_QUIT) //메시지가 winAPI종료가 아니라면

{

if (PeekMessage(&msg, NULL, 0, 0, PM\_REMOVE)) // 메시지가 있으면 처리

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

else //그렇지 않으면 에니메이션/게임 작업을 수행

{

}

}

// WM\_QUIT 메시지로 반환

return static\_cast<char>(msg.wParam);

}

//핸들값을 얻기 위해서

HWND WinAPI::GetHwnd()

{

return WinAPI\_hwnd;

}

//윈 프로시져

LRESULT WinAPI::WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

//메시지

switch (message)

{

case WM\_CREATE: //창이 만들어졌으면

{

// 윈도우 만들시 DirectX12Base 를 저장

LPCREATESTRUCT pCreateStruct = reinterpret\_cast<LPCREATESTRUCT>(lParam);

SetWindowLongPtr(hWnd, GWLP\_USERDATA, reinterpret\_cast<LONG\_PTR>(pCreateStruct->lpCreateParams));

}

return 0;

case WM\_KEYDOWN: //키버튼이 눌렸으면

return 0;

case WM\_KEYUP: //키버튼이 때어젔으면

return 0;

//case WM\_PAINT: //사용하지 않는 이유: 이함수는 다시 그리는용인데 run()함수에서 처리하고 있기 때문

// return 0;

case WM\_DESTROY: //파괴되었을때

PostQuitMessage(0);

return 0;

}

// 디폴트값 대신 모든 메시지 처리

return DefWindowProc(hWnd, message, wParam, lParam);

}

WinAPI.cpp(c++)

LRESULT WinAPI::WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

}

한동안 윈 프로시저 안에 메시지를 처리하는 switch문에서 break와 return을 사용하는 문제에 대해서 시끄러웠을 때가 있었습니다.

//윈 프로시져

LRESULT WinAPI::WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

//메시지

switch (message)

{

case WM\_CREATE: //창이 만들어졌으면

return 0;

case WM\_KEYDOWN: //키버튼이 눌렸으면

return 0;

case WM\_KEYUP: //키버튼이 때어젔으면

return 0;

//case WM\_PAINT: //사용하지 않는 이유: 이함수는 다시 그리는용인데 run()함수에서 처리하고 있기 때문

// return 0;

case WM\_DESTROY: //파괴되었을때

PostQuitMessage(0);

return 0;

}

// 디폴트값 대신 모든 메시지 처리

return DefWindowProc(hWnd, message, wParam, lParam);

}

위의 return 사용되신 아래 break사용으로 바꿀 수 있습니다.

//윈 프로시져

LRESULT WinAPI::WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

//메시지

switch (message)

{

case WM\_CREATE: //창이 만들어졌으면

break;

case WM\_KEYDOWN: //키버튼이 눌렸으면

break;

case WM\_KEYUP: //키버튼이 때어젔으면

break;

case WM\_DESTROY: //파괴되었을때

break;

default:

DefWindowProc(hWnd, message, wParam, lParam);

break;

return 0

}

return, break를 사용하는 입장의 차가 첨예하게 대립하는데 return의 경우는 함수의 종료를 신경 쓰고, break의 경우에는 switch의 종료를 신경 쓰는 편입니다. 개인적인 생각으로는 return, break 둘의 명확한 사용법만 알고 있으면 된다고 생각합니다 또한 switch 문에 하나로 같이 쓰지 않으면 된다고 생각합니다.

LPCREATESTRUCT pCreateStruct = reinterpret\_cast<LPCREATESTRUCT>(lParam);

원래 c를 쓰던 유저라 강제 캐스트를 c++스타일로 쓰는 걸 별로 좋아하지는 않습니다. 예전부터 코드작업을 했기에 c 스타일에 코드를 읽는데도 무리는 없다고 생각합니다. 하지만 c++ 스타일로 바꾸려는 것은 이 코드를 읽는 사람이 내가 아닌 다른 사람일 수도 있기 때문입니다. 연습하지 않으면 외워지지 않음으로 코드 연습 겸 정리합니다.  
명시적 타입 4가지  
1) static\_cast<> // 정적 캐스팅, 컴파일시 검사  
→ (c) float a = 1.0f / int b = (int)a;  
→ (c++) float a = 1.0f / int b = static\_cast<int>a;  
⇒ 값, 객체의 타입을 변환시킬 때 사용합니다.  
2) reinterpret\_cast<> // 포인트 안 데이터를 변환시킬 시  
ex)  
temp\* a = new Temp(0,0);  
unsigned int b = reinterpret\_cast<unsigned int>(a); //주소값으로 변경  
temp \*c = reinterpret\_cast<temp \*>(b); //다시 포인트로 변경  
⇒ 포인터 사이 형변환, 포인터와 변수로 변환시킬 시 사용합니다.  
3) const\_cast<> // const 함수 제거할 시  
⇒ 변경권이 없는 외부 DLL함수라던가 const 함수를 제거 할 시 사용  
4) dynamic\_cast<> // 런타임시 판별  
⇒ 포인터는 참조형을 캐스팅할 때만 사용가능. 호환되지 않는 자식형으로 캐스팅되면 NULL로 반환 RTTI(real time type information)을 사용하지 않으면 static\_cast<>와 동일 하다.  
c의 강제 캐스팅 ( ) 와 c++ 명시적 캐스팅의 차이점:  
구글링으로 확인해보면 가끔 c++이 안전하다고 적어 놓은 사이트들이 있는데, 토이프로젝트로 진행해보면 똑같이 런타임시 크래쉬 나는 경우가 대부분입니다. 따라서 안전성은 똑같다고 생각합니다. 하지만 c++의 장점은 프로젝트 검색할 때 ctrl + f 로 검색할 쉽게 검색하여 찾을 수 있고, 가독성이 좋다는 장점이 있습니다.  
(관련 글) <https://stackoverflow.com/questions/103512/why-use-static-castintx-instead-of-intx>

#include "stdafx.h"

#include "WinAPI.h" //다이렉트 x 파이프라인

\_Use\_decl\_annotations\_ //error c28213 해결 사용 이유: 정적분석 도구 에서 주석을 가져오도록 사용

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, int nCmdShow)

{

if (!WinAPI::Init( hInstance, nCmdShow)) //초기화 실패하면

return 0; // 0으로 리턴

return WinAPI::Run( ); //성공시 run 코드 실행

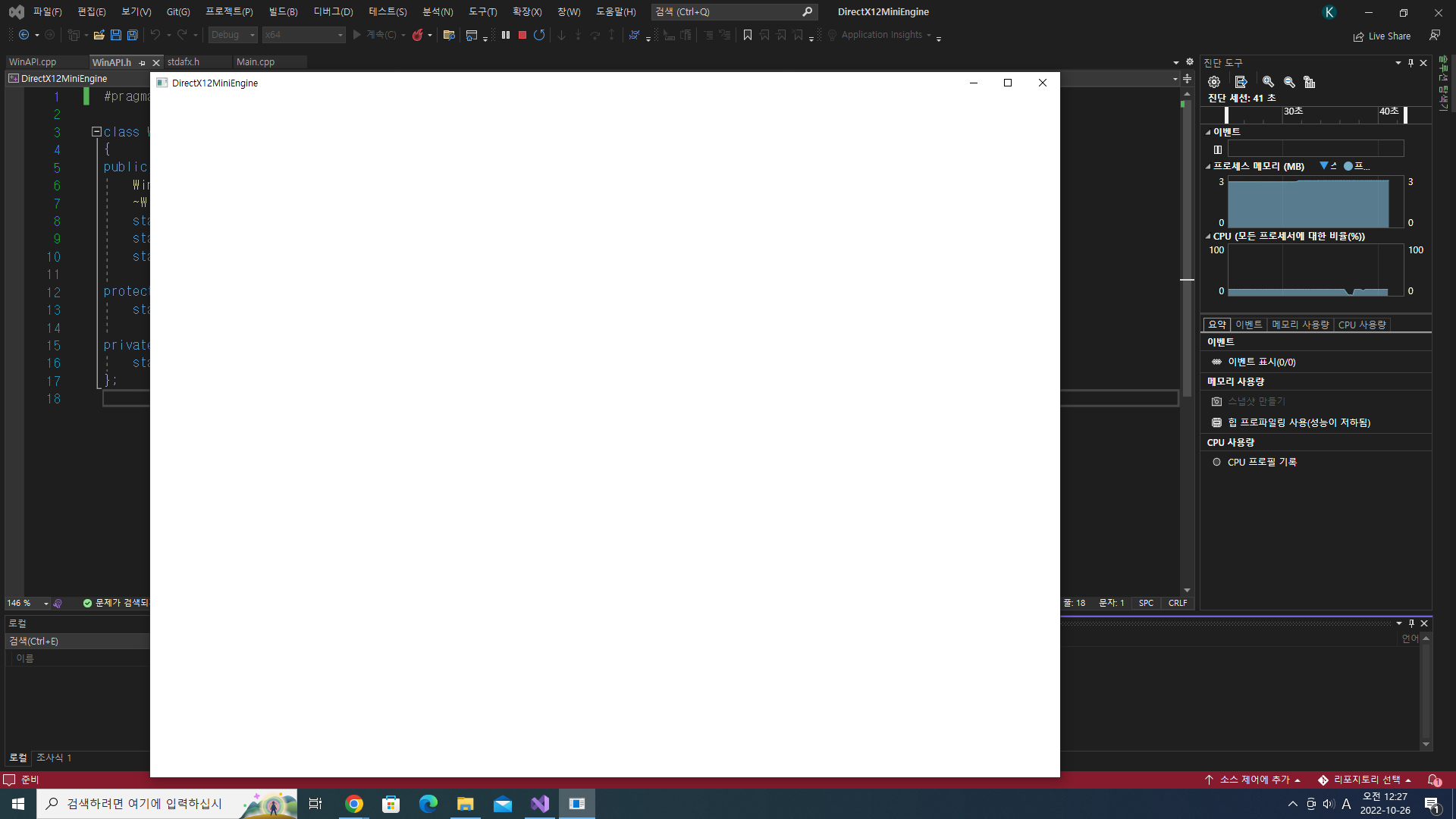
}

Main.cpp(c++)

\_Use\_decl\_annotations\_

**Warning C28213 해결**  
함수의 주석을 가저오도록 정적 코드 분석도구에 지시. 바이너리에 제공되는 API의 주석 중복 되어있는 것을 방지하기 위해서 사용

### 결과



## directX 초기화

### 이 파트를 정리하는 이유

→ 졸업과제에서는 점수를 위해 구현하기 바빠서 생각보다 하드웨어에 대해서 깊게 생각해 볼 수 없었다. 여유가 있을 때, 정리해 보기로 함

### 에러 및 해결 방법

c2102 error

→