09. Framework

개발/배포용 프로젝트 폴더 구성

① 빈 프로젝트 설정으로 새 솔루션 생성.
② 솔루션 내에 생성된 프로젝트 파일 에디터 내에서 제거.
③ 탐색기에서 프로젝트 위치 찾고 해당 위치에 Bin(Binary) 폴더와 include 폴더 생성
* Bin 폴더에는 실행 파일과 관련된 모든 파일, include 폴더에는 코드 파일과 관련된 모든 파일이 들어간다.
④ 프로젝트 파일들을 include 폴더로 옮기고, 에디터에서 솔루션을 클릭 – 기존 프로젝트를 추가해준다.
*왜 삭제하고 다시 추가해주는가?
솔루션은 프로젝트 생성/추가 시에 해당 프로젝트의 경로에 대한 정보를 지니고 있는데,
이 때 프로젝트 위치를 바꾸면 변경된 위치를 반영해 해당 정보를 수정해 주지는 않는다.
따라서 외부에서 프로젝트를 이동시켜 버리면 오류가 발생할 수 있다.
⑤ Bin 폴더에 실행 파일이 들어가도록 프로젝트 -설정의 출력 디렉토리에 모든 구성/플랫폼의 출력 디렉토리 위치를 Bin 폴더로 설정해준다.
*이때 디렉토리는 상대경로로 지정. (파일을 다른 곳에 저장하면 절대경로는 바뀌기때문.)
⑥ 각 구성/플랫폼 별로 대상 이름(Target name. 실행 파일 이름)을 설정해준다.

* 디버그로 프로젝트 빌드하면 디버깅을 위한 부가적인 코드도 내부적으로 다 돌아감.
따라서 실행 파일 용량이 커지고 속도도 느림. 따라서 배포시에는 Release로 빌드.

* 윈도우API를 활용한 wWinMain을 main entry로 사용할 경우 빌드시 오류가 날 수 있다.
이는 기본 Subsystem이 Console로 되어 있어 main() 함수를 찾기 때문이다.
따라서 이를 Windows로 변경해주어야 한다.

main.cpp

```cpp
#include <Windows.h>

int APIENTRY wWinMain( _In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR    lpCmdLine,
    _In_ int       nCmdShow )
{

    return 0;
}
```

macro.h

```cpp
#pragma once

#define SAFE_DELETE(p) if(p) { delete p; p = NULL; }
#define SAFE_DELETE_ARRAY(p) if(p) { delete[] p; p = NULL; }
```

game.h

```cpp
#pragma once

#include <Windows.h>
#include <list>
#include <vector>
#include <unordered_map>

using namespace std;

#include "Macro.h"
```

core.h

```cpp
#pragma once

#include "Game.h"

class CCore
{
private:
	static CCore*	m_pInst;

public:
	static CCore* GetInst()
	{
		if( !m_pInst )
			m_pInst = new CCore;
		return m_pInst;
	}

	static void DestroyInst()
	{
		SAFE_DELETE( m_pInst );
	}

private:
	CCore();
	~CCore();
};
```

Static 멤버 변수

1) 프로그램이 시작될 때 (클래스가 메모리에 로드 될 때) 변수가 만들어짐.
   객체 생성 시점이 아님.
2) 모든 클래스가 공유 함.

싱글톤 패턴

단 한 개의 인스턴스 만을 생성하여 사용하는 디자인 패턴.
여기서는 Core 클래스가 게임 전체를 관리하기 때문에
하나만을 생성하게 된다.

Core.cpp

```cpp
#include "Core.h"

CCore* CCore::m_pInst;

CCore::CCore()
{
}

CCore::~CCore()
{
}
```

static 멤버 변수는 전역 변수처럼 취급된다.
static 멤버 변수는 모든 객체가 공유해야 하므로 프로그램 전체 영역에서
해당 메모리 공간이 유지되야 하기 때문에 전역 범위 에서 초기화 해주어야 함.

또 static 이고 const 가 아닌 경우 클래스 선언 부에서 선언한 static
멤버 변수의 경우 선언을 했음에도 실체가 없고 그저 선언만 한 것이기
때문에 해당 변수에 접근하면 존재하지 않는 변수에 접근하는 것이므로
빌드시 에러가 발생한다.
따라서 변수가 존재함을 알려주기 위해 클래스 정의부에서 선언을 함으로써
컴파일러에게 알려 사용하는 것이다.