C# 기초

DELICIOUS
GAMES

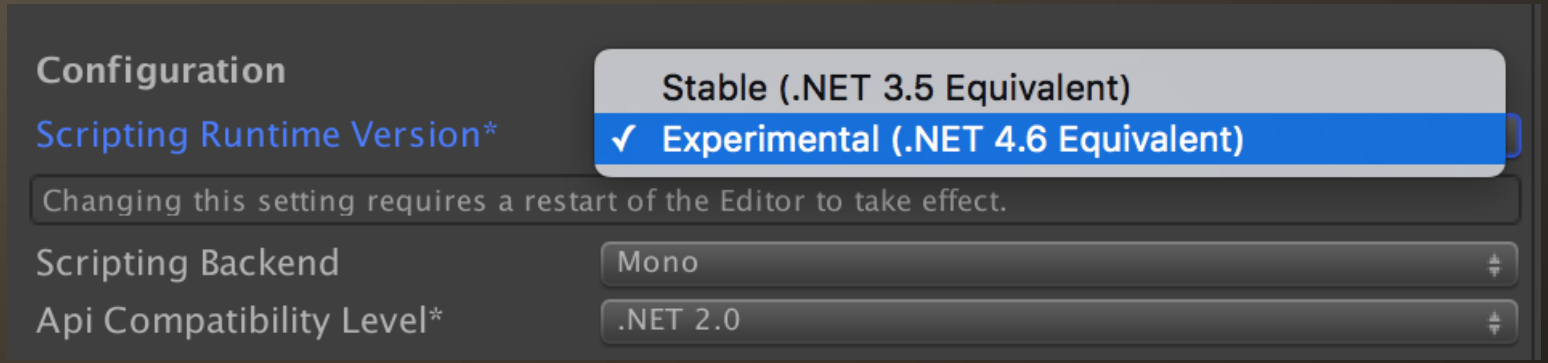
C# Scripting Basics

C# Script

- C#
  - MS에서 창조
  - C++의 객체지향성을 향상
  - 플랫폼 독립적인 프레임워크 기반
- Script
  - 컴파일 과정 없이 실행 → 인터프리팅
  - JIT 컴파일 (Just-in-Time Compilation)
    - 중간코드로 속도 향상

Unity 2017.1

- C#6, MONO/.NET 4.6 정식 지원



- 아직은 3.5가 안정적

참고 자료

- 유니티 에반젤리스트의 발표
  - https://www.slideshare.net/gukhwanji/c-26604507

- 유니티 C# 기초 강좌
  - http://itmining.tistory.com/26

형

DELICIOUS
GAMES

Types

역할

- 값의 종류를 구분
- 대입과 비교되는 값들은 서로 같은 형이어야 한다.
  (혹은 서로 변환 가능한 형이어야 한다.)

```
int x = 1;          // int형 변수에 값 1을 대입
char c = Ch;        // char형 변수에 값 Ch 변수의 값을 대입
char e = 1;         // char형 변수에 숫자값을 대입하면 오류
```

특성

- 숫자와 문자 외에도 다양한 형이 존재

```
bool x = true;
byte b = 255;
int i = 1;
float f = 0.5f;
double d = 8.15;
```

종류

종류	형	범위 / 설명
부호가 있는 정수	sbyte	-128 ~ 127, 1B
	short	-32,768 ~ 32,767, 2B
	int	-2,147,483,648 ~ 2,147,483,647, 4B
	long	-9,233,372,036,854,775,808 ~ 9,233,372,036,854,775,807, 8B
부호가 없는 정수	byte	0 ~ 255, 1B
	ushort	0 ~ 65,535, 2B
	uint	0 ~ 4,294,967,295, 4B
	ulong	0 ~ 18,446,744,073,709,551,615, 8B

종류

종류	형	범위 / 설명
부동소수점 수	`float`	$1.5 \times 10^{-45} \sim 3.4 \times 10^{38}$, 7자리수 유효, 4B
	`double`	$5.0 \times 10^{-324} \sim 1.7 \times 10^{308}$, 15자리수 유효, 8B
고정밀 소수점 수	`decimal`	$1.0 \times 10^{-28} \sim 7.9 \times 10^{28}$, 28자리수 유효, 12B
불린	`bool`	true, false
문자	`char`	유니코드 문자
문자열	`string`	유니코드 문자열
오브젝트	`object`	모든 형으로 저장할 수 있는 객체

형 추론

- C++11에서의 auto 키워드와 같은 기능
- 할당 받는 값의 형을 자동으로 추론하여 적용한다.
- var 형은 선언하는 동시에 초기화가 필요하다.

```
var a = true; // a는 bool형이 된다.
var b = 255;  // b는 int형이 된다.
var c = 0.5f; // c는 float형이 된다.
var d = 8.15; // d는 double형이 된다.
```

배열

- 같은 형의 값이 여러개 나열된 집합

```
int[] x = { 1, 2, 3, 4, 5 };
var y = new int[5];
var z = new[] { "a", "b", "c" };

int[,] m = { { 1, 0 }, { 0, 1 } };
float[,] g = new float[3,5];

string[][] s = new string[2][];
s[0] = new string[] { "월", "화", "수", "목", "금" };
s[1] = new string[] { "오전", "오후" };
```

연산자

DELICIOUS GAMES

Operators

역할

- 항을 어찌저찌하여 새로운 값을 만들어 낸다.
  - 이항 연산자 → a **x** b
  - 단항 연산자 → **x**a 혹은 b**x**

- 연산자 우선순위
  - 서로 섞여있을 경우 무엇을 먼저 계산할지 결정하는 순서

특성

- 계산 순서
  - ◯우선순위가 높은 연산자부터 먼저 계산한다.
  - ◯같은 우선순위는 왼쪽에서 오른쪽으로 계산한다.

```
var x = 1 + 2 + 3;              // 왼쪽에서 오른쪽으로 1+2 → 3+3 → 6
var y = 1 + 2 * 3;              // 곱하기 먼저 2*3 → 1+6 → 7
var z = 3 * (2 + 1);   // 괄호 먼저 2+1 → 3*3 → 9
Debug.Log("x = " + x);
Debug.Log("y = " + y);
Debug.Log("z = " + z);
```

종류

연산자	식	용도
.	x.y	멤버에 접근
()	f(x)	메서드 호출
[]	x[i]	배열
++, --	x++, x--	대입하고 1 증가/감소
+, -	+x, -x	숫자의 부호
!	!x	부정, x가 true이면 false, 아니면 true
~	~x	각각의 비트를 반전. 1011 --> 0100
++, --	++x, --x	1 증가/감소 시키고 대입
()	(T)x	형변환
*	x * y	곱셈
/	x / y	나눗셈

종류

연산자	식	용도
%	y % x	나머지
+	x + y	덧셈
-	x - y	뺄셈
<<	x << y	x를 y개수만큼 비트를 위로 이동
>>	x >> y	x를 y개수만큼 비트를 아래로 이동
<	x < y	x가 y보다 작으면 true
>	x > y	x가 y보다 크면 true
<=	x <= y	x가 y보다 작거나 같으면 true
>=	x >= y	x가 y보다 크거나 같으면 true
==	x == y	x와 y가 같으면 true
!=	x != y	x와 y가 다르면 true

종류

연산자	식	용도
&	x & y	x와 y의 비트 and (10 & 11 = 10)
^	x ^ y	x와 y의 비트 xor (10 ^ 11 = 01)
|	x | y	x와 y의 비트 or (10 | 11 = 11)
&&	x && y	x와 y가 둘 다 true이면 true
||	x || y	x와 y 중 하나만 true라면 true
??	x ?? y	x가 값이 있으면 x, 값이 없으면(null이면) y
? :	x ? y : z	x가 true면 y, 아니면 z
=	x = y	x에 y값을 대입
+=, -=, *=, /= %=, &=, |=, ^=	x += y	x에 x + y 값을 대입 (나머지도 동일 방식)
=>	() => {}	람다식

괄호의 활용

- 괄호는 모든 연산자보다 우선 처리
  - 연산자의 순위가 헷갈릴 수 있으므로
    의도에 맞게 괄호를 적절히 활용하는 것이 좋다.

```
var x = a & b || c ? a ^ c : a ^ b;                    // 의도가 헷갈림
var y = ((a & b) || c) ? (a ^ c) : (a ^ b);  // 순서가 명확
```

구문

DELICIOUS GAMES

Statement

구문 Statement

- 하나의 실행 단위
  - 문장의 완성
  - 반드시 세미콜론으로 끝나야 한다.

```
var x = 1 + 2 + 3;
var y = 1 + 2 * 3;
var z = 3 * (2 + 1);
```

스코프 Scope

- 유효 범위
  - 선언된 내용이 영향을 미치는 범위
  - 중괄호로 감싸서 표현

```
int a = 1;
if (a == 1)
{
        int b = 2;
}
Debug.Log("b = " + b);
```

선언문

- 변수나 상수를 정의하는 구문
  - 변수는 값을 자유롭게 대입할 수 있고,
    상수는 선언과 동시에 대입하여 변화 없이 사용한다.

```
int x;
x = 1;
const float pi = 3.14f;
```

연산문

- 값을 계산하는 구문
  - 계산된 값은 출력하거나 변수에 대입한다.

```
float area = pi * (x * x);
```

조건문

- 조건에 따라 선택하는 구문
  - 조건을 검사하고 선택적으로 실행한다.

```
if (x == 0)
        run();
else
        exit();
```

```
switch (x)
{
        case 0:
                run();
        break;
        default:
                exit();
```

반복문

- 조건에 따라 여러번 반복하는 구문
  - 조건을 검사하고 반복 실행한다.

```
do
{
    x *= 2;
} while (i < 10);

while (i < 10)
{
    x *= 2;
}
```

```
for (int i = 0; i < 10; ++i)
{
    x *= 2;
}

int[] y = { -1, 1, 2, -2, 0 };
foreach (int x in y)
{
    Debug.Log(x);
}
```

제어문

- 흐름을 제어하는 구문
  - 현재 스코프에서 빠져나가거나 특정 구문으로 이동한다.
  - goto, return 키워드도 이에 해당
    - goto
    - return

```
while (i > 0)
{
    if (i % 2 == 1)
    {
        Debug.Log("홀수");
        continue;
    }
    Debug.Log("짝수");

    ++i;
    if (i > 10)
        break;
}
```

예외 처리문

- 예외적인 상황을 선별해서 처리하는 구문
  - 스코프 내부에서 발생한 예외를 감지한다.

```
try {
  if (x < 0)
    throw
      new IndexOutOfRangeException(
        "범위를 벗어남");
  run(x);
}
catch (Exception e) {
  Debug.Log(e);
}
finally {
  x = 0;
}
```

그밖의 구문들

- checked, unchecked 문
  - 수치 연산에서 오버플로우 발생 여부를 확인한다.
  - checked
- yield 문
  - 처리를 잠시 미루고 다음에 계속한다.
  - yield

그밖의 구문들

- fixed 문
  - 가비지 컬렉터가 변수를 재배치하는 것을 방지한다.
  - fixed
- lock 문
  - 상호배타적인 잠금을 관리한다.
  - lock

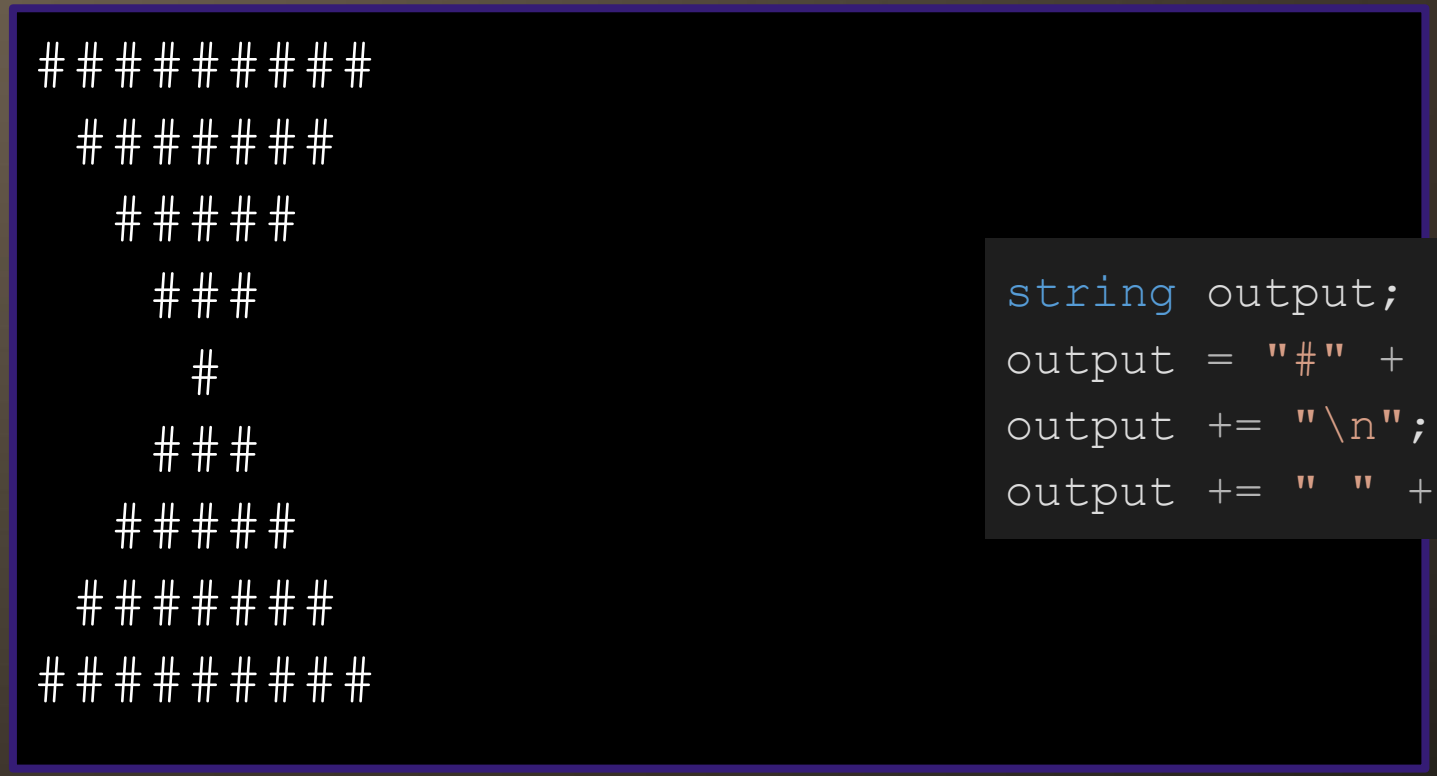
그밖의 구문들

- using 문
  - 리소스를 사용하는 스코프를 정의한다.
  - using

함수

- 함수
  - 구문들을 모아서 한번에 실행되도록 만든 덩어리

```
void boo() { }
int step(int n) { return n + 5; }
bool run(object obj)
{
    if (obj == null)
        return false;
    return obj.work();
}
```

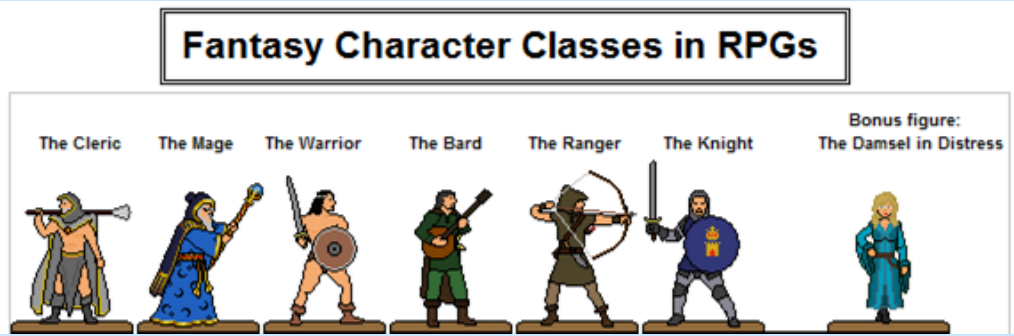



```
string output;
output = "#" + "#";
output += "\n";
output += " " + "#";
```

실습1: 유니티 콘솔창에 위와 같이 출력해 봅시다. (string 활용하여 한번에 출력)

실습2: 인스펙터에서 int를 입력 받아 위의 출력 개수를 조정해 봅시다.

클래스



DELICIOUS
GAMES

Class

클래스

- 관련된 것들을 모아서 멤버로 관리
  - 변수
  - 함수

```
class Person
{
    public string name;
    bool run();
}
```

멤버 사용

- 선언된 클래스 개체를 생성해서 객체로 만들고 .연산자를 통해 접근

```
Person girl = new Person();
girl.name = "유니티짱";
girl.run();
```

셀프 참조

- this 키워드를 통해서 자신의 멤버를 참조
  - 생략 가능하지만, 혼동될 경우 사용

```
class Person
{
        public string name;
        public int age;

        Person(int age)
        {    this.age = age; }
}
```

멤버

- 클래스에 소속된 함수와 변수, 상수들
  - 생성자 / 소멸자
  - 변수 / 상수
  - 메서드
  - 프로퍼티
  - 이벤트 / 인덱서
  - 연산자
  - 중첩 선언

접근지시자

- 멤버에 접근할 수 있는 권한 설정
  - `public`
    - 내외부에서 자유롭게 접근
  - `protected`
    - 해당 개체와 하위 개체(상속 또는 포함)만 접근 가능
  - `private`
    - 해당 개체의 멤버들만 접근 가능
  - `internal`
    - 같은 코드 내부에서만 접근 가능
  - `internal protected`, `protected internal`
    - 같은 코드에 있더라도 해당 개체와 하위 개체만 접근 가능

멤버 변수 (필드)

- 클래스 개체에 소속된 변수
  - 클래스 내부의 정보 저장을 담당한다.

```
class Car {
        public float speed;
        bool engine =
false;
}
```

멤버 상수

- 클래스 개체에 소속된 상수
  - 고정된 값을 제공한다.
    - 옵션이나 부가값 들을 표현할 때

```
class Car {
        public float speed;
        bool engine = false;
        const float MAX_SPEED =
5.0f;
}
```

메서드

- 클래스 개체에 소속된 함수
  - 멤버들을 자유롭게 사용할 수 있다.

```
class Car {
    public float speed = 0;
    bool engine = false;

    void start() { engine = true; }
    void stop()  { speed = 0; engine = false; }

    bool work() {
        if (!engine) return false;
        speed = 5.0f;
        return true;
    }

    public bool run() {
        start();
        if (work()) return true;
        stop();
        return false;
    }
}
```

생성자

- 객체가 생성될 때 실행되는 함수
  - 초기화 등을 수행한다.

```
class Car {
        public float speed;
        bool engine;

    Car(float speed)
    {
        this.speed = speed;
        engine = (speed != 0);
    }
}

Car mycar = new Car(1.5f);
```

소멸자

- 객체가 소멸될 때 실행되는 함수
  - 정리 작업 등을 수행 한다.

```
class Car {
        public float speed;
        bool engine;

        ~Car()
        {
                stop();
        }

        void start() { engine = true; }
        void stop() {
                speed = 0;
                engine = false;
        }
```

프로퍼티

- 클래스 밖에서는 변수처럼 안에서는 함수처럼 사용하는 메서드
  - 클래스 내부의 복잡한 작업을 외부에 단순화 시켜 제공한다.

```
class Person
{
        private string name;
        public string Name { get; private set; }
}
```

프로퍼티

```
class Person
{
        private string name;
        public string Name
        {
                get {
                        return name;
                }

                set {
                        if (value.Length > 0)
                                name = value;
                }
        }
}
```

프로퍼티

- 유니티의 Inspector에서 프로퍼티를 나타나도록 사용하려면 직렬화가 필요

```
class Person
{
    [SerializeField]
    private string name;
    public string Name { get; private set; }
}
```

인덱서

- 클래스 밖에서 배열처럼 사용하는 프로퍼티

```
class Car {
        private string[] type = { "경차", "승용차", "승합차", "버스", "트럭" };
        public string this[int I] {
                get {
                        if (I < 0 || I >= type.Length)
                                return "알 수 없음";
                        return type[I];
                }
                set {
                        if (I < 0 || I >= type.Length)
                                return;
                        type[I] = value;
                }
        }
```

인덱서

- 클래스 밖에서 배열처럼 사용하는 프로퍼티

```
class Car {
      private string[] type = { … };
      public string this[int I] { … }
}
Car mycar = new Car();
Debug.Log(mycar[2]);      // "승합차" 출력
```

정적 멤버

- 객체로 생성하지 않아도 미리 생성되어 있고 사용할 수 있는 멤버
  - static 키워드로 선언
  - 클래스 이름으로 직접 접근

```
class Car
{
        public static string count;
}
Car.count = 1;
```

정적 멤버

- 정적 멤버는 일반 멤버에게 접근이 불가능
  - 일반 멤버는 정적 멤버를 자유롭게 사용할 수 있지만
    정적 멤버는 오직 정적 멤버만 사용 가능

```
class Car
{
    public float speed = 0;
    public static string count;
    public static void ListUp()
    {
        speed = 1.0f;
        this.speed = 2.5f;
        count++;
    }
}
```

상속

- 이미 만들어진 클래스를 이어받아서 차이점만 추가로 구현
  - 공통된 멤버들을 부모 클래스로 모으면 코드의 양을 줄이고 유지보수에 용이하며 개체간 성격을 분류할 수 있다.

상속

부모-자식 클래스

```
public class Car
{
        public float speed = 0;
        public Car(float s)
        {   speed = s;  }
}
public class Truck : Car
{
        public float load = 0;
        public Truck(float s, float l) : base(s)
        {   load = l;   }
}
```

상속

부모-자식 클래스

```
public class Truck : Car
{
        public float load = 0;
        public Truck(float s, float l) : base(s)
        {    load = l;    }
        public bool loadPackage(float weight) {
                if (weight > load) return false;
                speed -= 0.5f;
                return true;
        }
}
```

상속

- 부모-자식 클래스
  - 자식은 부모에게 할당할 수 있지만
  - 부모는 자식에게 할당할 수 없다.

```
Car a = new Car(1.5f);
Truck b = new Truck(1.0f, 5.0f);
a = b;
b = a;
```

오버라이딩

- 부모에게 이미 있는 멤버를 자식이 변경
  - 전혀 다른 결과를 반환할 수 있다.

```
public class Car {
        virtual public string getName()
        {    return "자동차";    }
}

public class Truck : Car {
        override public string getName()
        { return "트럭"; }
}
```

오버라이딩

- 부모에게 이미 있는 멤버를 자식이 변경
  - 전혀 다른 결과를 반환할 수 있다.

```
Car a = new Car();
Truck b = new Truck();
Debug.Log(a.getName()); // 자동차
Debug.Log(b.getName()); // 트럭

a = b;
Debug.Log(a.getName()); // 트럭
Debug.Log(b.getName()); // 트럭
```

추상화

- 추상 클래스
  - 내용이 완전히 결정되지 않은 클래스

- 추상 메서드
  - 내용이 비어있는 메서드
  - 앞으로 구현될 것이라 기대
  - 상속을 통해 구체적으로 구현

추상화

- 내용이 없는 빈 메서드 선언
  - 자식 클래스가 오버라이딩 하여 구현

```
abstract public class Car {
    abstract public string getName();
}
public class Truck : Car {
        override public string getName()
        { return "트럭"; }
}
public class SUV : Car {
        override public string getName()
        { return "스포츠유틸리티비클"; }
}
```

추상화

- 추상 클래스는 인스턴스 생성 불가능

```
Car a = new Car(); // 에러
Car a;
a = new Truck();
Debug.Log(a.getName()); // 트럭
a = new SUV();
Debug.Log(a.getName()); // 스포츠유틸리티비클
```

인터페이스

- 전체 추상화 클래스
  - 내용이 아무것도 없는 클래스
  - 메서드 이름과 형식만 선언 가능
  - 모든 선언이 public
- 상속 조건
  - 인터페이스를 상속한 자식 인터페이스 불가능
  - 인터페이스는 여러개 중복 상속 가능

인터페이스

```
interface ICar {
        string GetName();
}
interface IEngine {
        string
GetEngType();
}
```

```
public class Truck : ICar, IEngine {
        public string GetName()
        { return "트럭"; }
        public string GetEngType()
        { return "경유"; }
}
public class SUV : ICar, IEngine {
        public string GetName()
        { return "스포츠유틸리티비클"; }
        public string GetEngType()
        { return "휘발유"; }
}
```

```
public class Person
{
    public string name;              // 이름
    virtual public string Work()     // 행동 출력
    { return "데굴데굴"; }
}
interface ILife
{
    string GetSex();     // 성별 출력
}

public class Girl : Person, ILife {
    public Girl()
    { }
    override public string Work()
    { }
    string GetSex()
    { }
}
```

실습3: Person 클래스를 부모로 ILife 인터페이스를 상속하는 Girl, Boy, Baby 클래스를 구현해 봅시다.

고급진 주제

DELICIOUS
GAMES

High Class

구조체

- 참조 형식 Reference Type
  - 실제 값이 들어있는 메모리를 가르킨다.
  - C#에서 생성된 배열과 클래스 객체는 참조 전달

- 값 형식 Value Type
  - 실제 값을 복사한다.
  - 배열을 제외한 기본형과 구조체는 값 복사

구조체

pass by reference

cup = 

fillCup( )

pass by value

cup = 

fillCup( )

www.penjee.com

구조체

- 클래스와 같이 멤버를 가지는 구조체

```
public struct StructValue
{
    public const string type = "구조체";
    public int x;
}

public class ClassReference
{
    public const string type = "클래스";
    public int x;
}
```

구조체

대입시 값을 복제

```
StructValue    s = new StructValue();
ClassReference c = new ClassReference();
s.x = 1;     c.x = -1;

StructValue    si = s;
ClassReference ci = c;
si.x++;      ci.x--;

Debug.Log("struct = " + s.x + ", class = " + c.x);       // 1. -2
Debug.Log("struct = " + si.x + ", class = " + ci.x);     // 2, -2
```

복제형

- 정수형
  - sbyte, short, int, long, byte ...
- 소수형
  - float, double, decimal
- 문자형, 불린형
  - char, bool
- 열거형
  - enum
- 구조체
  - struct
- null

참조형

- 문자열, 배열
  - string, []
- 클래스, 인터페이스
  - class, interface
- 객체
  - object
- 함수 대리자
  - delegate

이름 공간 Namespace

- 여러 개체의 패키지를 묶는 개념
  - 클래스, 구조체, 소스 등을 이름 하나로 묶음
  - 외부에서는 using 키워드로 통채로 사용 가능

```
namespace UnityChan
{
    public class Person { }
    public class Car { }
}

UnityChan.Car mycar = new UnityChan.Car();

using UnityChan;
Car mycar = new Car();
```

형변환 Casting

- 서로 호환되지 않는 형식을 변환하여 대입

```
int a = 123;
double x;
x = a;          // 더 큰 개념으로 자동 변환
a = x;          // 에러
a = (int)x; // 소수점을 버리고 정수로 변환하여 대입
```

형변환 Casting

- 서로 호환되지 않는 형식을 변환하여 대입

```
public class Person {}
public class Girl : Person {}

Person p = new Person();
Girl g = new Girl();
p = g;
g = p;
g = (Girl)p;
```

형변환 Casting

as 키워드로 변환

```
public class Person {}
public class Girl : Person {}

Person p = new Person();
Girl g = new Girl();
g = (Girl)p;     // 변환이 불가능하면 예외 발생
g = p as Girl;   // 변환이 불가능하면 null을 대입
```

제너릭 Generic

- 동일한 로직인데 형만 다를 때 일반화
  - generic

```
public void Swap(ref int x, ref int y)
{ var t = x; x = y; y = t; }
public void Swap(ref string x, ref string y)
{ var t = x; x = y; y = t; }

public void Swap<T>(ref T x, ref T y)
{ var t = x; x = y; y = t; }
```

컬렉션 Collection

- 객체 집합을 관리하는 방법
  - 배열과 유사하지만 배열의 크기는 정적인 반면 컬렉션은 크기가 유동적이다.

```
var nameArr = new string[2];
nameArr[0] = "철수";
nameArr[1] = "영희";

var nameColl = new List<string>();
nameColl.Add("철수");
nameColl.Add("영희");
nameColl.Insert(1, "희철");
```

컬렉션 Collection

- Dictionary<TKey, TValue>
- HashSet<T>
- LinkedList<T>
- List<T>
- Queue<T>
- Stack<T>

foreach 반복문

- 집합 내부의 모든 원소에 순서대로 접근
  - 반복문에서 컬렉션을 사용할 때는 주의

```
var names = new[] { "철수", "영희" };
foreach (var n in names)
{
    Debug.Log(n);
}
```

IEnumerable 제너릭 인터페이스

- 반복문이 가능한 컬렉션을 구현
  - 읽기 전용의 안전한 컬렉션도 적용 가능

```
static readonly IEnumerable<string> names
                                    = new[] { "철수", "영희", "희철" };

foreach (var n in names)
    Debug.Log(n);
```

yield와 코루틴

- 코루틴 Coroutine
  - 유니티에서 제공하는 기능
  - 코드의 진행을 분해하여 중간중간 실행을 함으로써 멀티쓰레딩과 유사한 동시성을 확보
  - 실질적인 성능 개선 방안 중 하나
  - 목적에 따라 가독성 향상
- 참고 자료
  - gamecodingschool.org
  - doyouknowunity.blogspot.kr

함수 대리자

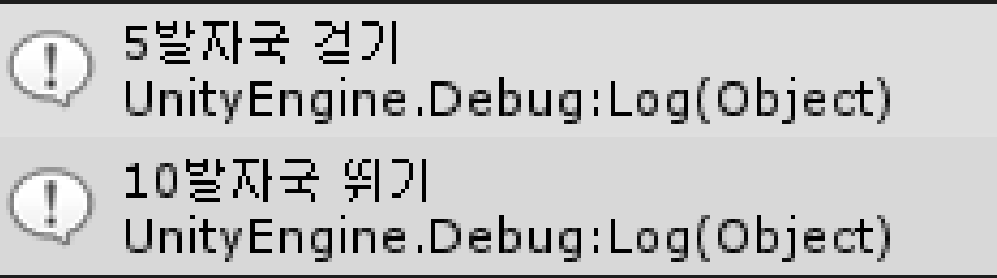
- 메서드가 대입이 가능하도록 참조

```
delegate void Move(int step);

void Walk(int step) {
      Debug.Log(step + "발자국 걷기");
}
void Run(int step) {
      Debug.Log(step + "발자국 뛰기");
}
```

함수 대리자

- 메서드가 대입이 가능하도록 참조

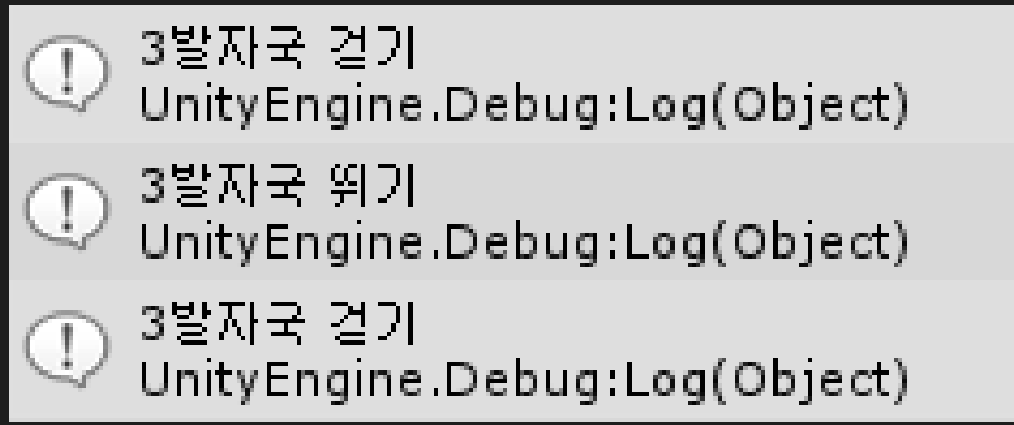
```
Move m;

m = Walk;
m(5);

m = Run;
m(10);
```



함수 대리자

- 메서드가 대입이 가능하도록 참조

```
Move m;


m = Walk;
m += Run;
m += Walk;
m(3);
```



함수 대리자

- 미리 정의된 대리자
  - 리턴 값이 없으면 Action<T>
  - 리턴 값이 있으면 Func<T>

```
Action<int> m;


m = Walk;
m(2);
```

람다식 Lambda Expression

- 일회용 함수
  - 선언없이 바로 사용하는 함수

```
int Sqrt(int x) { return x * x; }
Debug.Log( Sqrt(8) );

Func<int, int> sqrt;
sqrt = (x => x * x);
Debug.Log(sqrt(8));
```

확장 메서드 Extension Method

- 기존의 클래스를 전혀 수정하지 않고 확장

```
public static class ExString {
    public static int GetExLength(this string val) {
        return val.Length + 2;
    }
}

Debug.Log(ExString.GetExLength("유니티짱"));
Debug.Log("유니티짱".GetExLength());
```

LINQ

- 컬렉션을 SQL처럼 쿼리해서 사용하자

```
var arr = new[] { 5, 6, 3, 7, 2, 8 };
List<int> selC = new List<int>();
foreach (var x in arr) {
    if (x % 2 == 1)
        selC.Add(x);
}

var selA = from x in arr where x % 2 == 1 select x;
var selF = arr.Where(x => x % 2 == 1);
```

LINQ

- 컬렉션을 SQL처럼 쿼리해서 사용하자

```
var arr = new[] { 3, 5, 6, 3, 7, 2, 8, 1, 1 };
var selF = arr.Where(x => x % 2 == 1) // { 3, 5, 3, 7, 1, 1 }
                              .Distinct()                  // { 3, 5,
7, 1 }
                              .OrderBy(x => x);            // { 1, 3,
5, 7 }
```

LINQ

표준 쿼리 연산자	반환 형식	즉시 실행	지연된 스트리밍 실행	지연된 비스트리밍 실행
Aggregate	TSource	X		
All	Boolean	X		
Any	Boolean	X		
AsEnumerable	IEnumerable<T>		X	
Average	단일 숫자 값	X		
Cast	IEnumerable<T>		X	
Concat	IEnumerable<T>		X	
Contains	Boolean	X		
Count	Int32	X		
DefaultIfEmpty	IEnumerable<T>		X	
Distinct	IEnumerable<T>		X	
ElementAt	TSource	X		

LINQ

표준 쿼리 연산자	반환 형식	즉시 실행	지연된 스트리밍 실행	지연된 비스트리밍 실행
ElementAtOrDefault	TSource	X		
Empty	IEnumerable<T>	X		
Except	IEnumerable<T>		X	X
First	TSource	X		
FirstOrDefault	TSource	X		
GroupBy	IEnumerable<T>			X
GroupJoin	IEnumerable<T>		X	X
Intersect	IEnumerable<T>		X	X
Join	IEnumerable<T>		X	X
Last	TSource	X		
LastOrDefault	TSource	X		
LongCount	Int64	X		

LINQ

표준 쿼리 연산자	반환 형식	즉시 실행	지연된 스트리밍 실행	지연된 비스트리밍 실행
Max	단일 숫자 값, TSource 또는 TResult	X		
Min	단일 숫자 값, TSource 또는 TResult	X		
OfType	IEnumerable<T>		X	
OrderBy	IOrderedEnumerable<TElement>			X
OrderByDescending	IOrderedEnumerable<TElement>			X
Range	IEnumerable<T>		X	
Repeat	IEnumerable<T>		X	
Reverse	IEnumerable<T>			X
Select	IEnumerable<T>		X	
SelectMany	IEnumerable<T>		X	
SequenceEqual	Boolean	X		
Single	TSource	X		

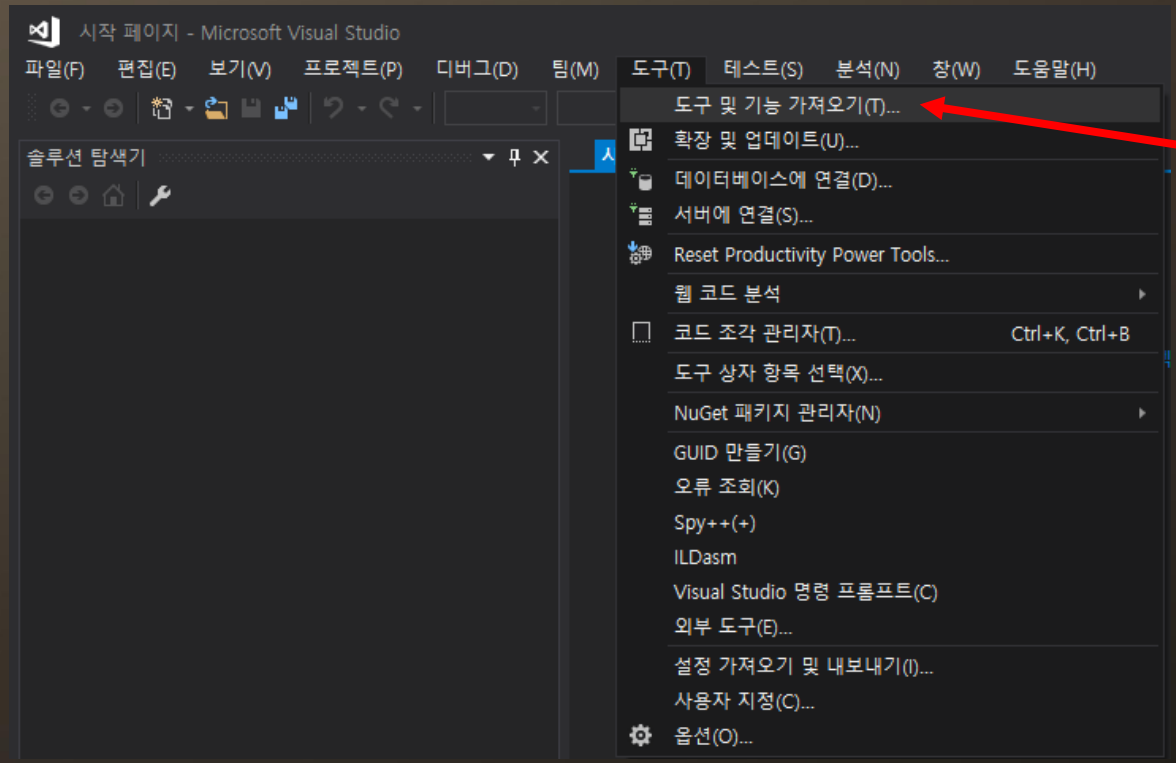
LINQ

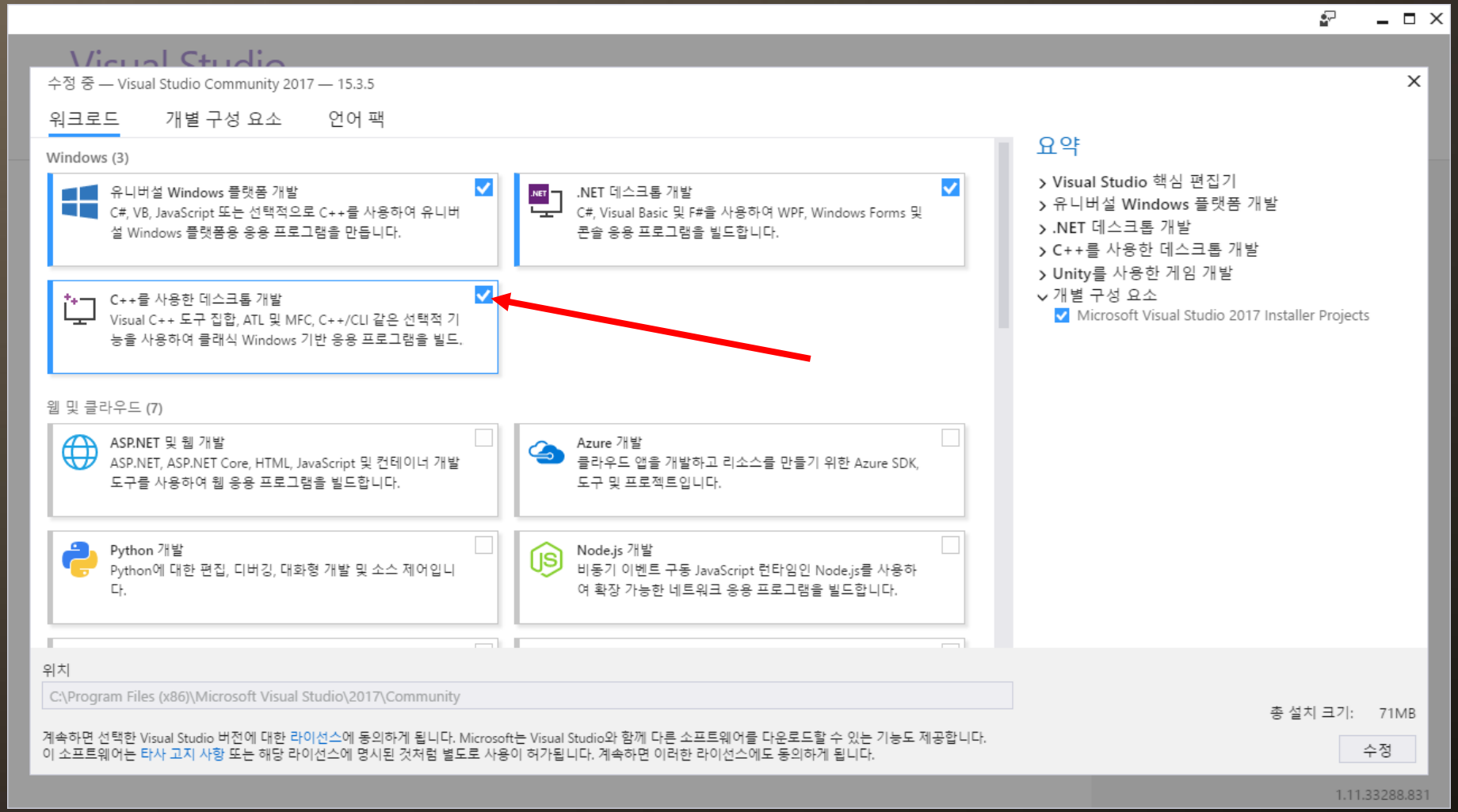
표준 쿼리 연산자	반환 형식			
SingleOrDefault	TSource	X		
Skip	IEnumerable<T>		X	
SkipWhile	IEnumerable<T>		X	
Sum	단일 숫자 값	X		
Take	IEnumerable<T>		X	
TakeWhile	IEnumerable<T>		X	
ThenBy	IOrderedEnumerable<TElement>			X
ThenByDescending	IOrderedEnumerable<TElement>			X
ToArray	TSource 배열	X		
ToDictionary	Dictionary<TKey,TValue>	X		
ToList	IList<T>	X		
ToLookup	ILookup<TKey,TElement>	X		
Union	IEnumerable<T>		X	
Where	IEnumerable<T>		X	

외부 라이브러리 호출 (C++)

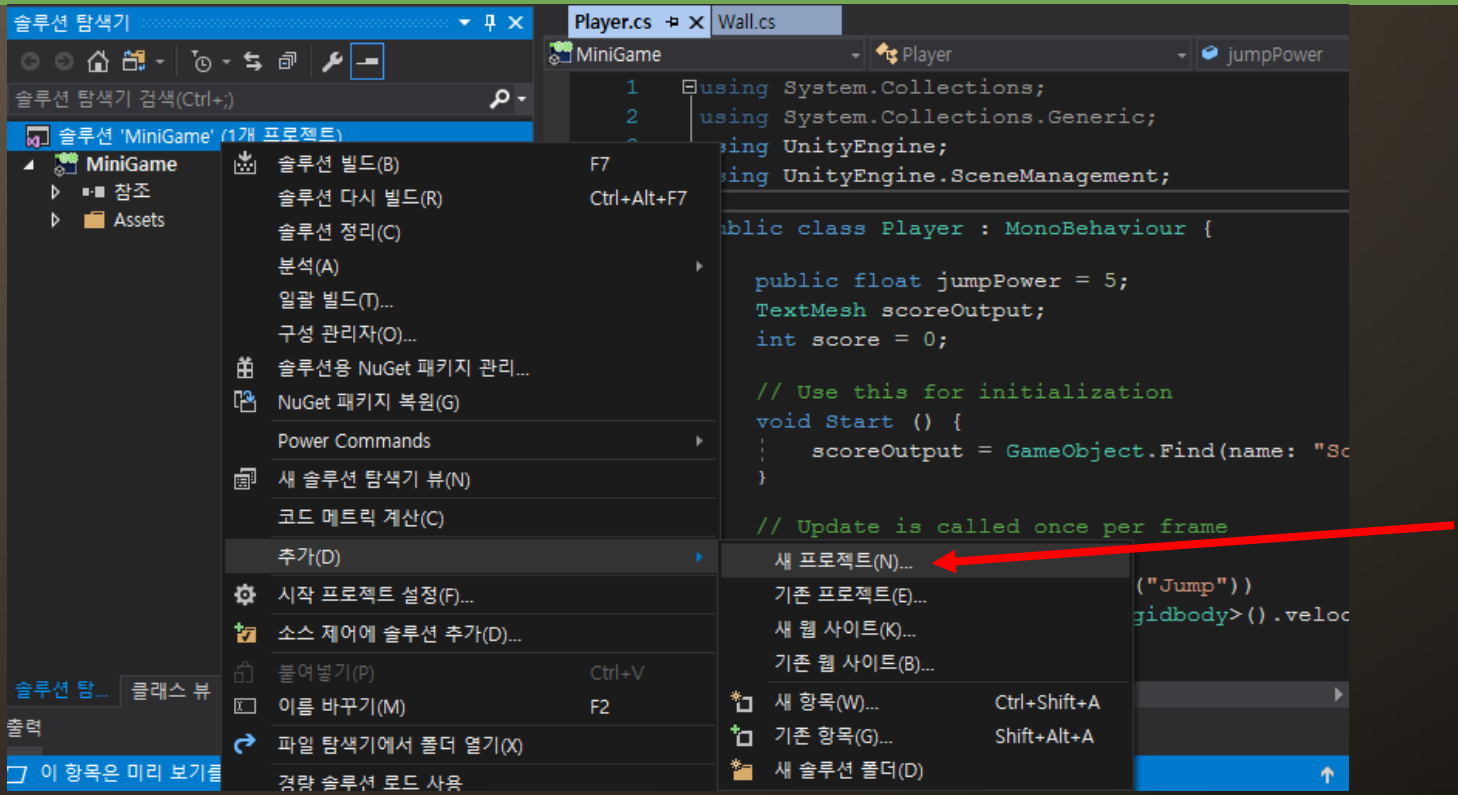
- C++로 DLL 만들기
  1. 새 프로젝트 추가
  2. Visual C++ > Windows 데스크톱 마법사
  3. 동적 연결 라이브러리
  4. 코드 작성
  5. 빌드 (Assets 아래의 Plugins 폴더로)
- 유니티 스크립트에서 사용

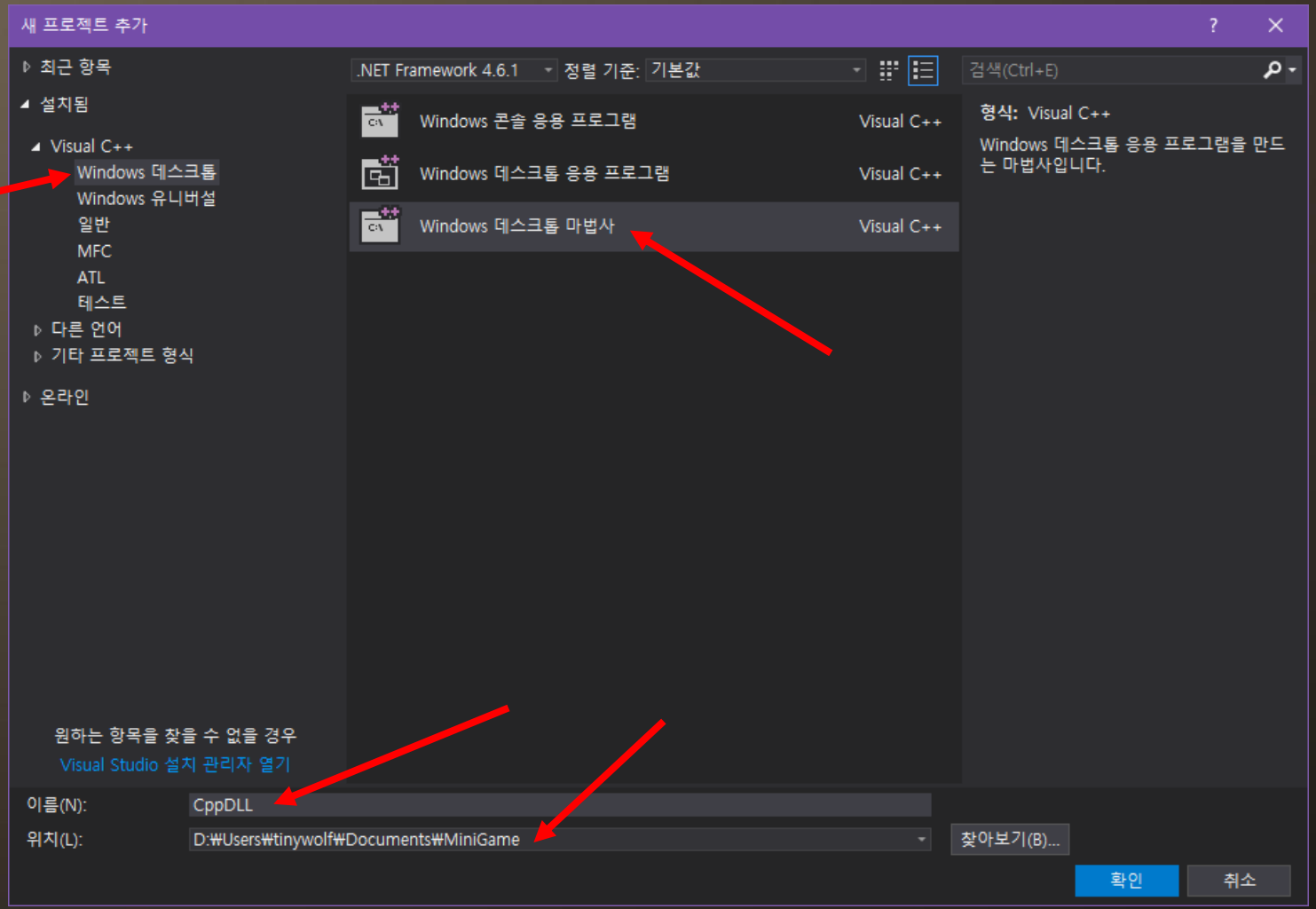
개발 도구 설치

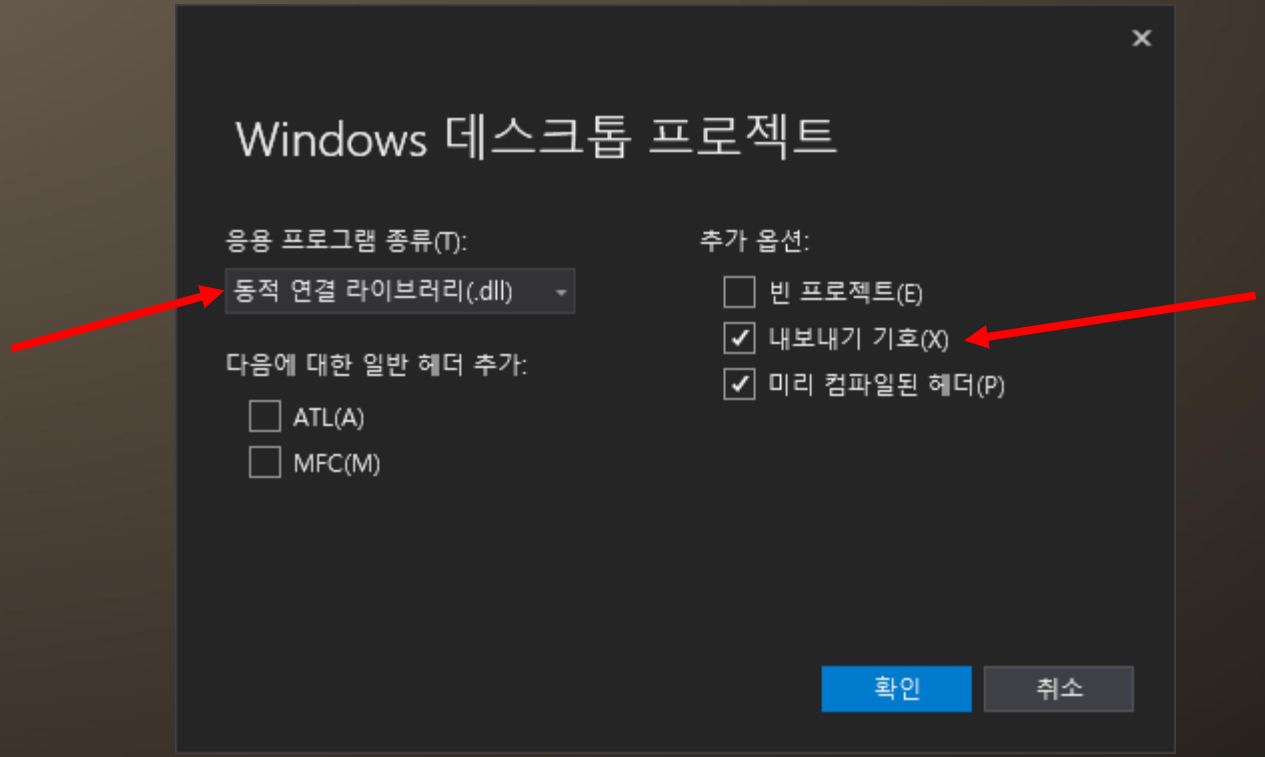

시작 페이지 - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 디버그(D) 팀(M) 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H)
솔루션 탐색기
도구 및 기능 가져오기(T)...
확장 및 업데이트(U)...
데이터베이스에 연결(D)...
서버에 연결(S)...
Reset Productivity Power Tools...
웹 코드 분석
코드 조각 관리자(T)... Ctrl+K, Ctrl+B
도구 상자 항목 선택(X)...
NuGet 패키지 관리자(N)
GUID 만들기(G)
오류 조회(K)
Spy++(+)
ILDasm
Visual Studio 명령 프롬프트(C)
외부 도구(E)...
설정 가져오기 및 내보내기(I)...
사용자 지정(C)...
옵션(O)...


수정 중 — Visual Studio Community 2017 — 15.3.5
워크로드
개별 구성 요소
언어 팩
Windows (3)
유니버설 Windows 플랫폼 개발
C#, VB, JavaScript 또는 선택적으로 C++를 사용하여 유니버설 Windows 플랫폼용 응용 프로그램을 만듭니다.
.NET 데스크톱 개발
C#, Visual Basic 및 F#을 사용하여 WPF, Windows Forms 및 콘솔 응용 프로그램을 빌드합니다.
C++를 사용한 데스크톱 개발
Visual C++ 도구 집합, ATL 및 MFC, C++/CLI 같은 선택적 기능을 사용하여 클래식 Windows 기반 응용 프로그램을 빌드.
웹 및 클라우드 (7)
ASP.NET 및 웹 개발
ASP.NET, ASP.NET Core, HTML, JavaScript 및 컨테이너 개발 도구를 사용하여 웹 응용 프로그램을 빌드합니다.
Azure 개발
클라우드 앱을 개발하고 리소스를 만들기 위한 Azure SDK, 도구 및 프로젝트입니다.
Python 개발
Python에 대한 편집, 디버깅, 대화형 개발 및 소스 제어입니다.
Node.js 개발
비동기 이벤트 구동 JavaScript 런타임인 Node.js를 사용하여 확장 가능한 네트워크 응용 프로그램을 빌드합니다.
요약
Visual Studio 핵심 편집기
유니버설 Windows 플랫폼 개발
.NET 데스크톱 개발
C++를 사용한 데스크톱 개발
Unity를 사용한 게임 개발
개별 구성 요소
Microsoft Visual Studio 2017 Installer Projects
위치
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
계속하면 선택한 Visual Studio 버전에 대한 라이선스에 동의하게 됩니다. Microsoft는 Visual Studio와 함께 다른 소프트웨어를 다운로드할 수 있는 기능도 제공합니다.
이 소프트웨어는 타사 고지 사항 또는 해당 라이선스에 명시된 것처럼 별도로 사용이 허가됩니다. 계속하면 이러한 라이선스에도 동의하게 됩니다.
총 설치 크기: 71MB
수정
1.11.33288.831

새 프로젝트 추가


솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'MiniGame' (1개 프로젝트)
MiniGame
참조
Assets
솔루션 탐...
클래스 뷰
출력
이 항목은 미리 보기를
솔루션 빌드(B) F7
솔루션 다시 빌드(R) Ctrl+Alt+F7
솔루션 정리(C)
분석(A)
일괄 빌드(T)...
구성 관리자(O)...
솔루션용 NuGet 패키지 관리...
NuGet 패키지 복원(G)
Power Commands
새 솔루션 탐색기 뷰(N)
코드 메트릭 계산(C)
추가(D)
시작 프로젝트 설정(F)...
소스 제어에 솔루션 추가(D)...
붙여넣기(P) Ctrl+V
이름 바꾸기(M) F2
파일 탐색기에서 폴더 열기(X)
경량 솔루션 로드 사용
새 프로젝트(N)...
기존 프로젝트(E)...
새 웹 사이트(K)...
기존 웹 사이트(B)...
새 항목(W)... Ctrl+Shift+A
기존 항목(G)... Shift+Alt+A
새 솔루션 폴더(D)
Player.cs
Wall.cs
MiniGame
Player
jumpPower
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Player : MonoBehaviour {
public float jumpPower = 5;
TextMesh scoreOutput;
int score = 0;
// Use this for initialization
void Start () {
scoreOutput = GameObject.Find(name: "Sc
}
// Update is called once per frame
("Jump"))
gidbody>().veloc


새 프로젝트 추가
최근 항목
설치됨
Visual C++
Windows 데스크톱
Windows 유니버설
일반
MFC
ATL
테스트
다른 언어
기타 프로젝트 형식
온라인
원하는 항목을 찾을 수 없을 경우
Visual Studio 설치 관리자 열기
.NET Framework 4.6.1
정렬 기준: 기본값
검색(Ctrl+E)
Windows 콘솔 응용 프로그램 Visual C++
Windows 데스크톱 응용 프로그램 Visual C++
Windows 데스크톱 마법사 Visual C++
형식: Visual C++
Windows 데스크톱 응용 프로그램을 만드는 마법사입니다.
이름(N): CppDLL
위치(L): D:₩Users₩tinywolf₩Documents₩MiniGame
찾아보기(B)...
확인
취소


Windows 데스크톱 프로젝트
응용 프로그램 종류(T):
동적 연결 라이브러리(.dll)
다음에 대한 일반 헤더 추가:
ATL(A)
MFC(M)
추가 옵션:
빈 프로젝트(E)
내보내기 기호(X)
미리 컴파일된 헤더(P)
확인
취소

외부 라이브러리 호출

- 헤더 파일 (CppDLL.h)

```
#ifdef CppDLL_EXPORTS
#define CPPDLL_API __declspec(dllexport)
#else
#define CPPDLL_API __declspec(dllimport)
#endif

extern "C" {
	CPPDLL_API const char* listDir(const char* szTarget);
}
```

외부 라이브러리 호출

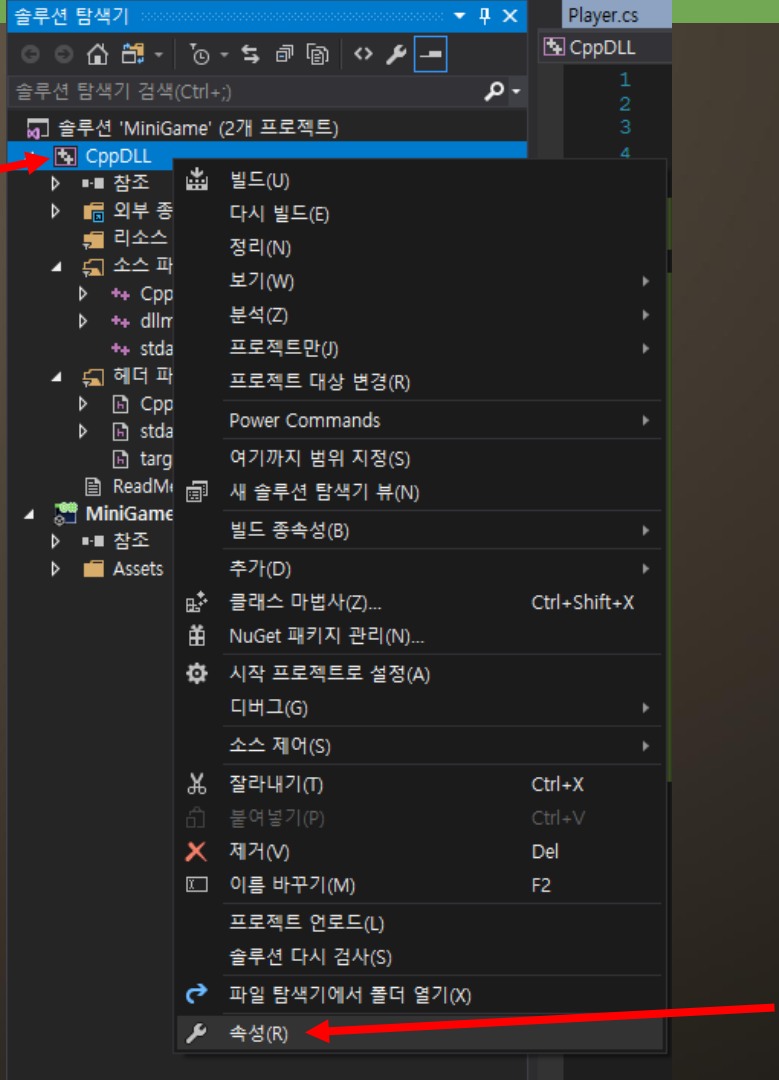
- 소스 파일 (CppDLL.cpp)

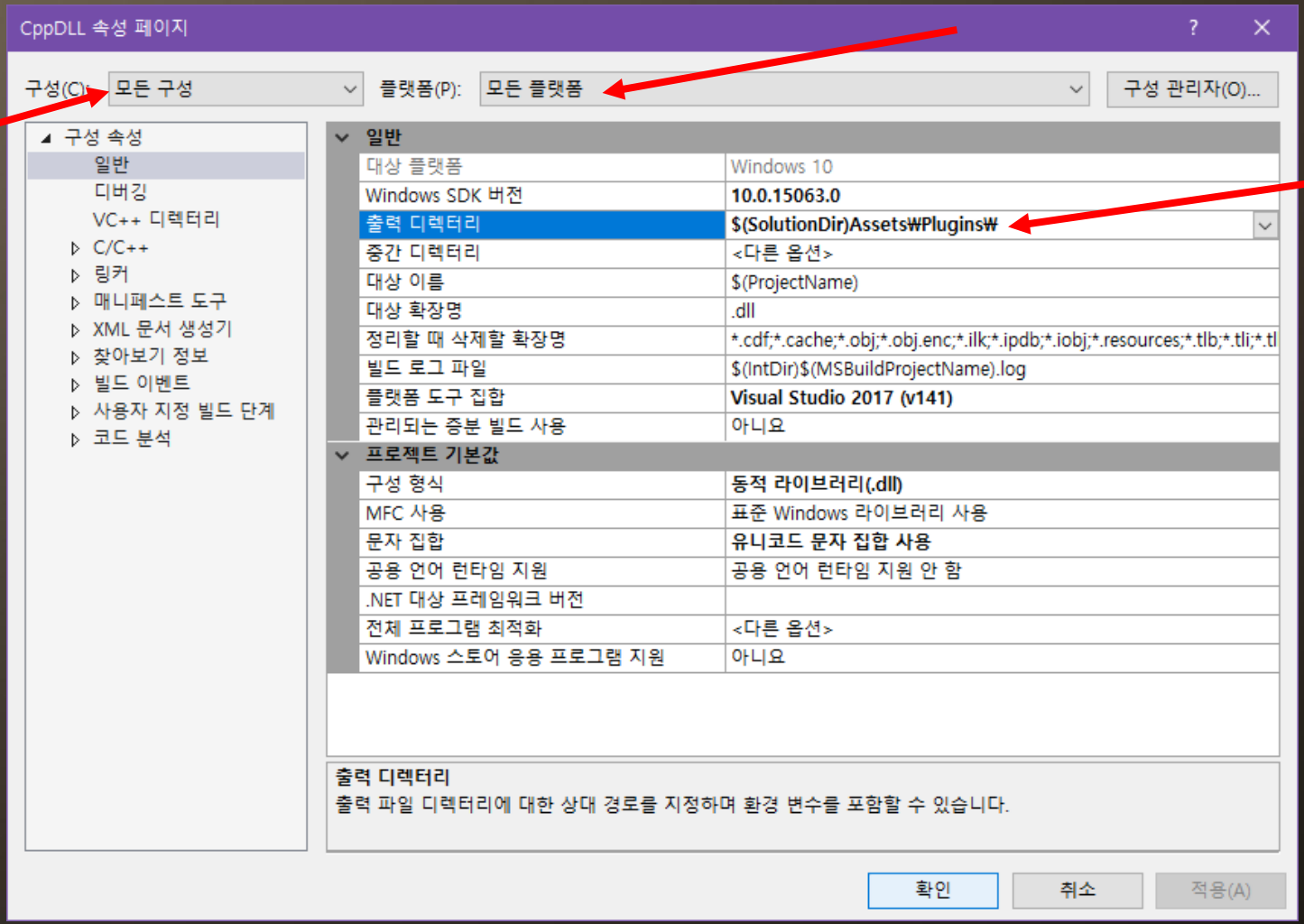
```
#include "stdafx.h"
#include "CppDLL.h"

#define BUFF_SIZE 1024

CPPDLL_API const char* listDir(const char* szTarget)
{
        static char szBuffer[BUFF_SIZE] = { 0 };

        WIN32_FIND_DATAA data;
        HANDLE hFind = FindFirstFileA(szTarget, &data);
```

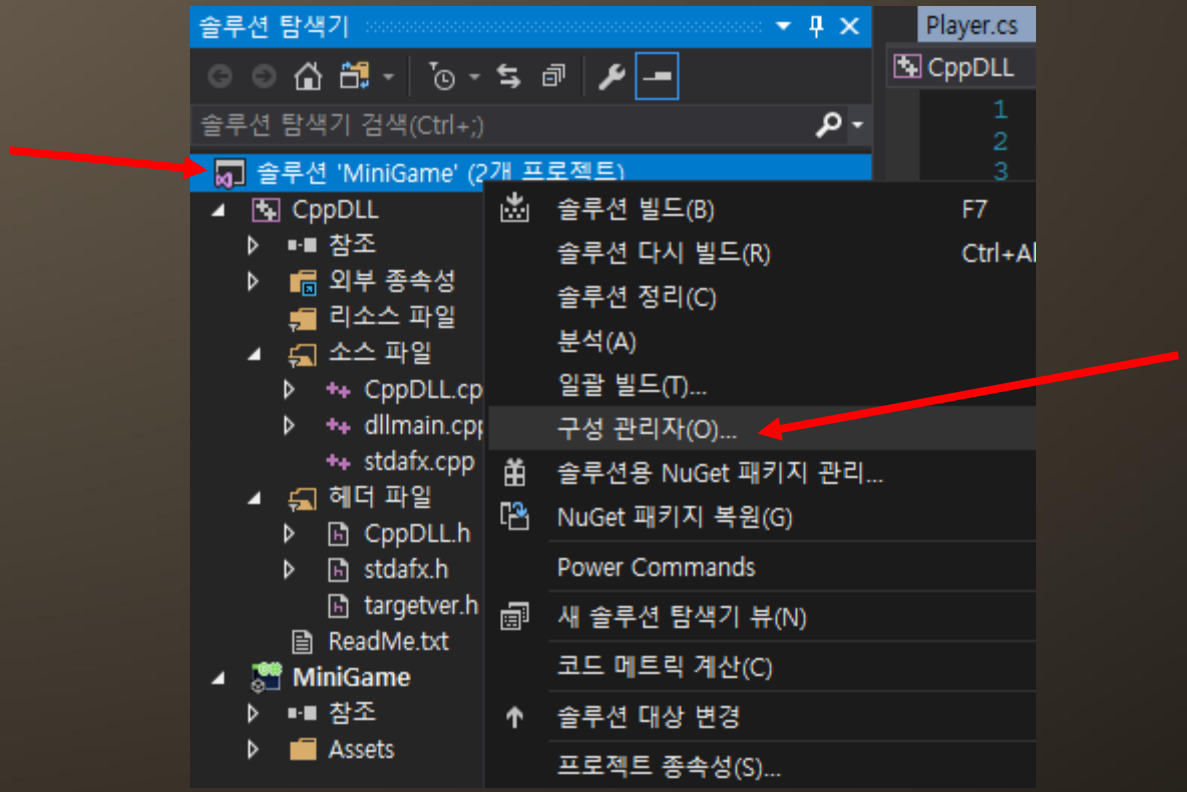
외부 라이브러리 호출

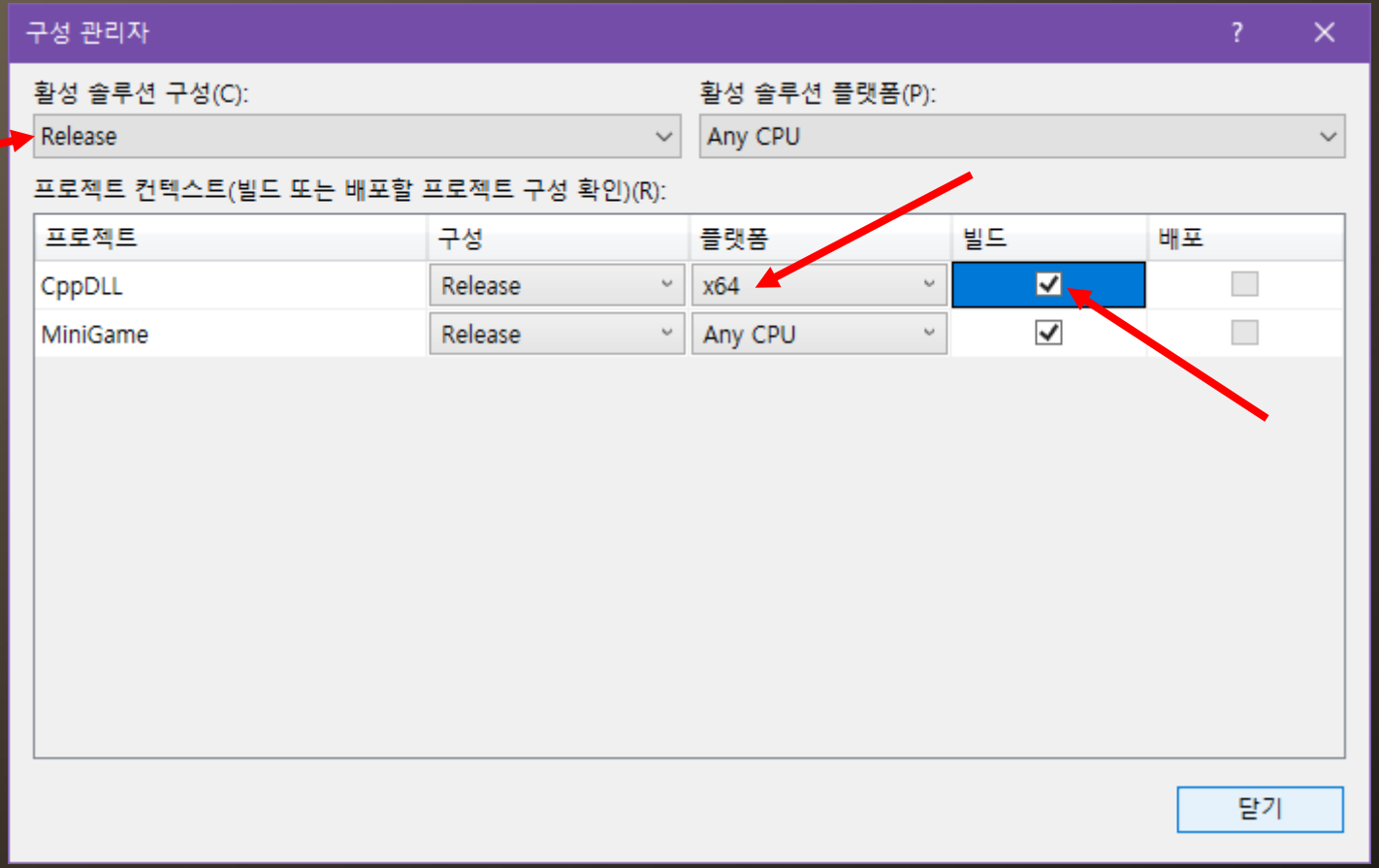
소스 파일 (CppDLL.cpp)

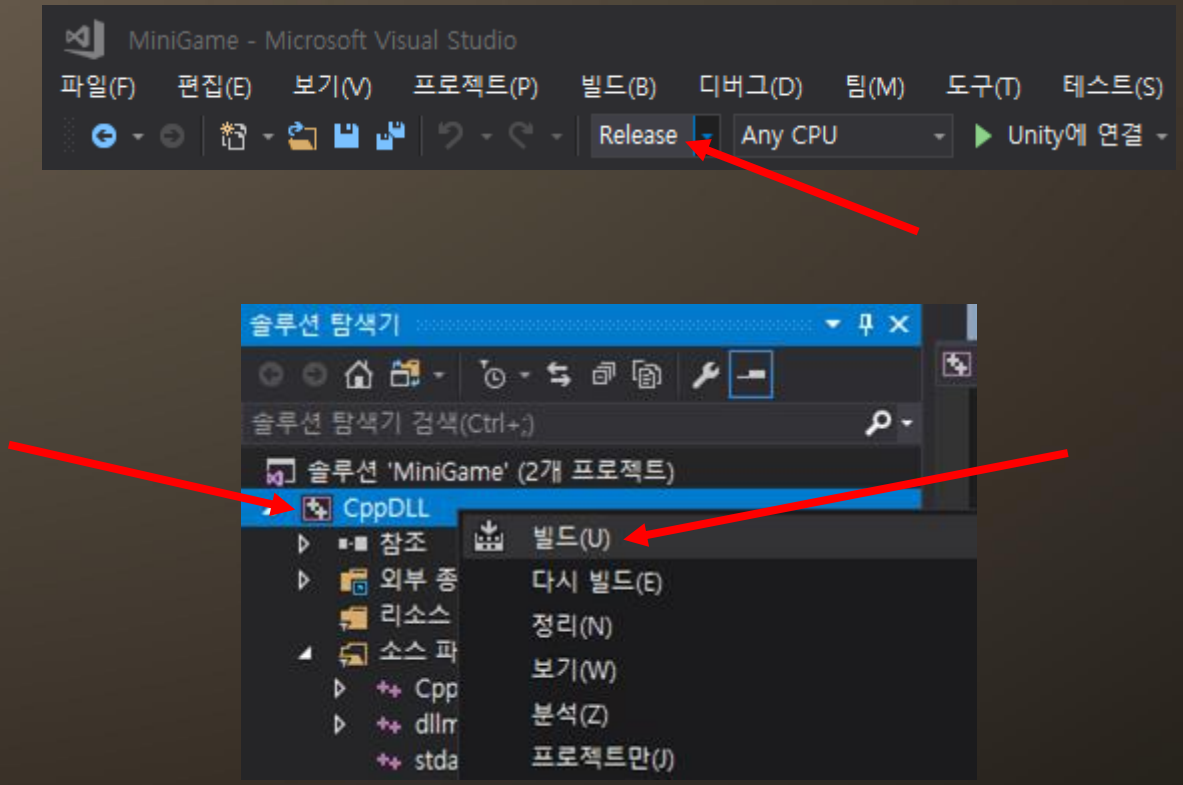
```
    if (hFind != INVALID_HANDLE_VALUE) {
            do {
                    strcat_s(szBuffer, BUFF_SIZE, data.cFileName);
                    strcat_s(szBuffer, BUFF_SIZE, "\n");
            } while (FindNextFileA(hFind, &data));
            FindClose(hFind);
    }

    return szBuffer;
}
```

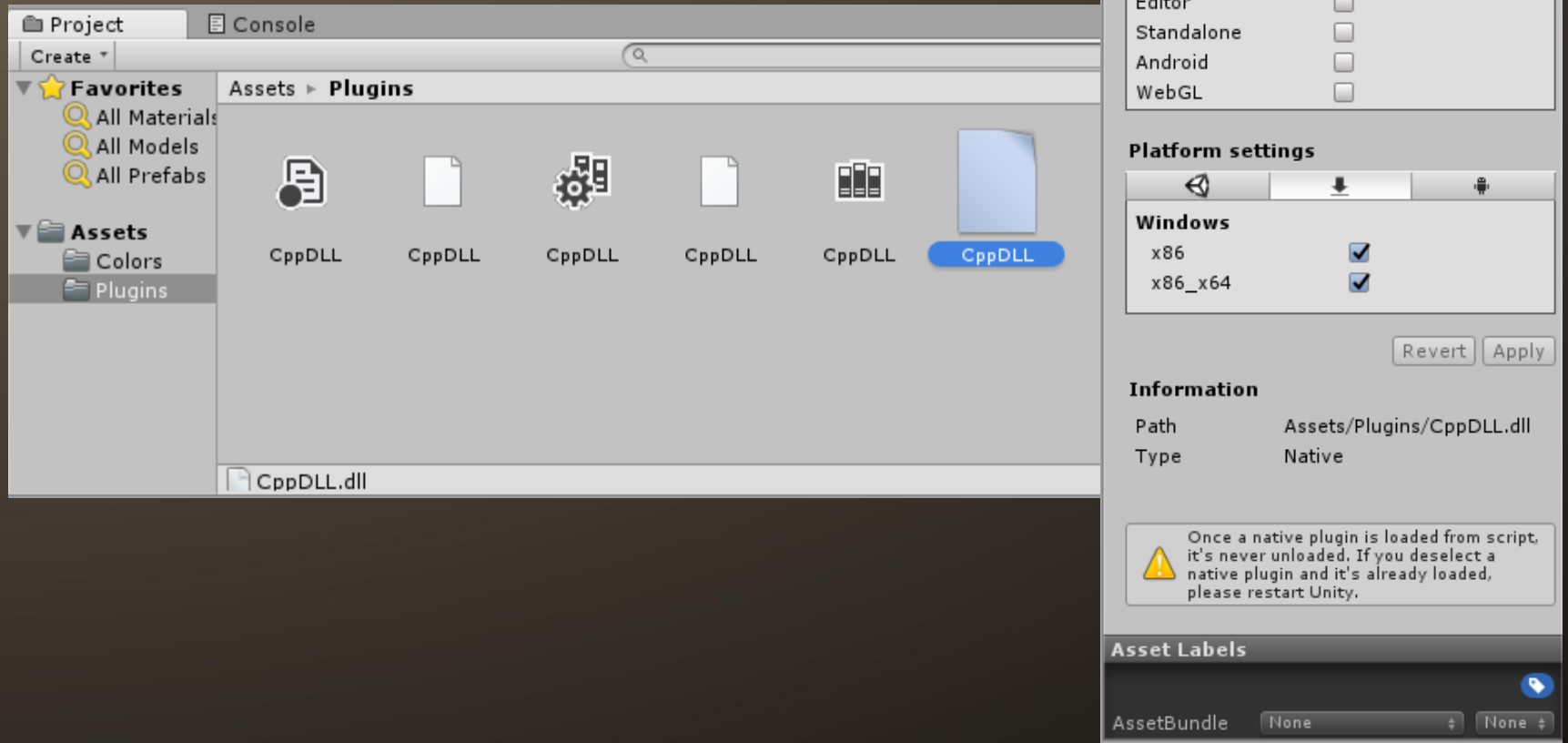

솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'MiniGame' (2개 프로젝트)
CppDLL
참조
외부 종
리소스
소스 파
Cpp
dllm
stda
헤더 파
Cpp
stda
targ
ReadM
MiniGame
참조
Assets
빌드(U)
다시 빌드(E)
정리(N)
보기(W)
분석(Z)
프로젝트만(J)
프로젝트 대상 변경(R)
Power Commands
여기까지 범위 지정(S)
새 솔루션 탐색기 뷰(N)
빌드 종속성(B)
추가(D)
클래스 마법사(Z)... Ctrl+Shift+X
NuGet 패키지 관리(N)...
시작 프로젝트로 설정(A)
디버그(G)
소스 제어(S)
잘라내기(T) Ctrl+X
붙여넣기(P) Ctrl+V
제거(V) Del
이름 바꾸기(M) F2
프로젝트 언로드(L)
솔루션 다시 검사(S)
파일 탐색기에서 폴더 열기(X)
속성(R)
Player.cs
CppDLL
1
2
3
4


CppDLL 속성 페이지
구성(C): 모든 구성
플랫폼(P): 모든 플랫폼
구성 관리자(O)...
구성 속성
일반
디버깅
VC++ 디렉터리
C/C++
링커
매니페스트 도구
XML 문서 생성기
찾아보기 정보
빌드 이벤트
사용자 지정 빌드 단계
코드 분석
일반
대상 플랫폼 Windows 10
Windows SDK 버전 10.0.15063.0
출력 디렉터리 $(SolutionDir)Assets₩Plugins₩
중간 디렉터리 <다른 옵션>
대상 이름 $(ProjectName)
대상 확장명 .dll
정리할 때 삭제할 확장명 *.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.tl
빌드 로그 파일 $(IntDir)$(MSBuildProjectName).log
플랫폼 도구 집합 Visual Studio 2017 (v141)
관리되는 증분 빌드 사용 아니요
프로젝트 기본값
구성 형식 동적 라이브러리(.dll)
MFC 사용 표준 Windows 라이브러리 사용
문자 집합 유니코드 문자 집합 사용
공용 언어 런타임 지원 공용 언어 런타임 지원 안 함
.NET 대상 프레임워크 버전
전체 프로그램 최적화 <다른 옵션>
Windows 스토어 응용 프로그램 지원 아니요
출력 디렉터리
출력 파일 디렉터리에 대한 상대 경로를 지정하며 환경 변수를 포함할 수 있습니다.
확인
취소
적용(A)


솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'MiniGame' (2개 프로젝트)
CppDLL
참조
외부 종속성
리소스 파일
소스 파일
CppDLL.cp
dllmain.cp
stdafx.cpp
헤더 파일
CppDLL.h
stdafx.h
targetver.h
ReadMe.txt
MiniGame
참조
Assets
솔루션 빌드(B) F7
솔루션 다시 빌드(R) Ctrl+Al
솔루션 정리(C)
분석(A)
일괄 빌드(T)...
구성 관리자(O)...
솔루션용 NuGet 패키지 관리...
NuGet 패키지 복원(G)
Power Commands
새 솔루션 탐색기 뷰(N)
코드 메트릭 계산(C)
솔루션 대상 변경
프로젝트 종속성(S)...
Player.cs
CppDLL
1
2
3


구성 관리자
활성 솔루션 구성(C):
Release
활성 솔루션 플랫폼(P):
Any CPU
프로젝트 컨텍스트(빌드 또는 배포할 프로젝트 구성 확인)(R):
프로젝트
구성
플랫폼
빌드
배포
CppDLL
Release
x64
MiniGame
Release
Any CPU
닫기

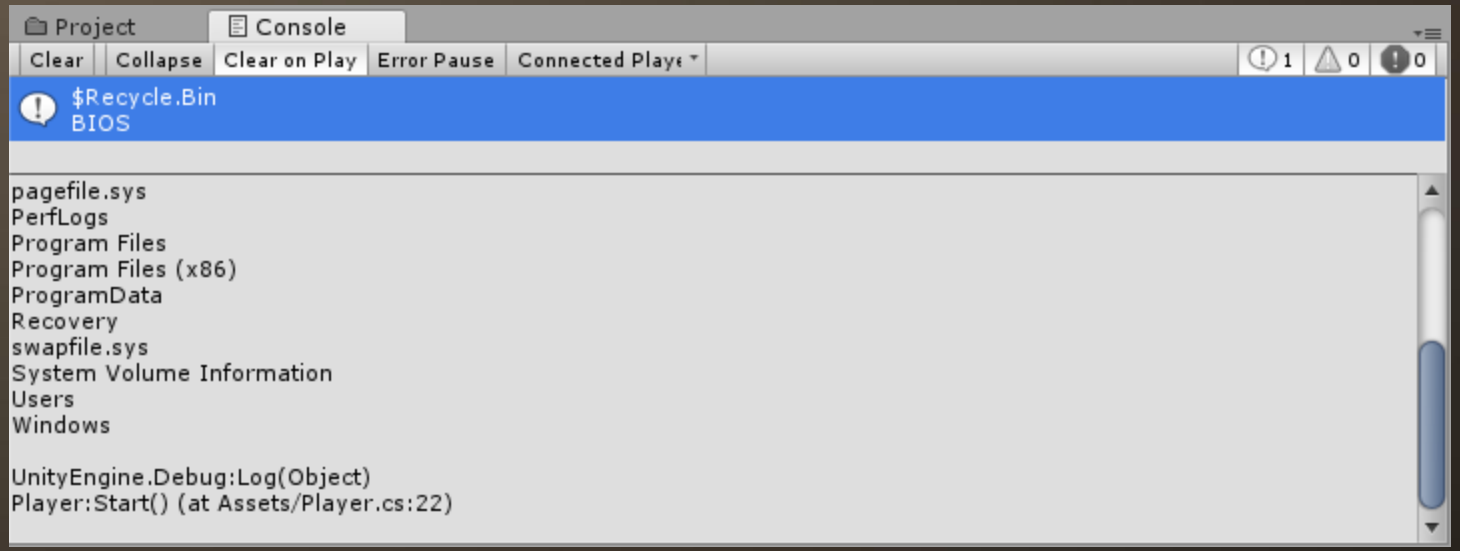

MiniGame - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 도구(T) 테스트(S)
Release
Any CPU
Unity에 연결
솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'MiniGame' (2개 프로젝트)
CppDLL
참조
외부 종
리소스
소스 파
Cpp
dllm
stda
빌드(U)
다시 빌드(E)
정리(N)
보기(W)
분석(Z)
프로젝트만(J)


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Colors
Plugins
Assets ▸ Plugins
CppDLL
CppDLL
CppDLL
CppDLL
CppDLL
CppDLL
CppDLL.dll
Inspector
Services
CppDLL Import Settings
Open
Select platforms for plugin
Any Platform
Exclude Platforms
Editor
Standalone
Android
WebGL
Platform settings
Windows
x86
x86_x64
Revert
Apply
Information
Path
Assets/Plugins/CppDLL.dll
Type
Native
Once a native plugin is loaded from script, it's never unloaded. If you deselect a native plugin and it's already loaded, please restart Unity.
Asset Labels
AssetBundle
None
None

외부 라이브러리 호출

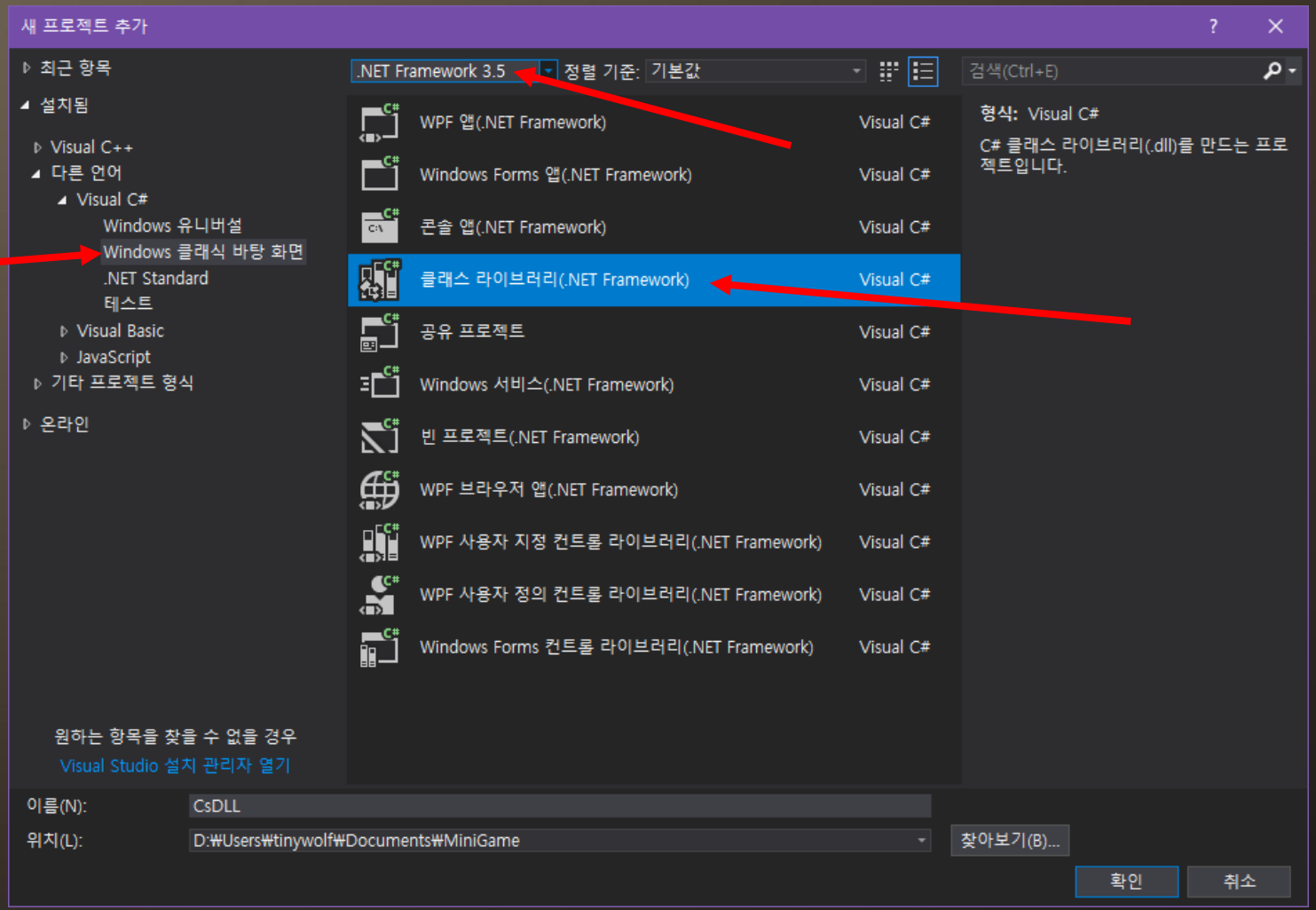
- 유니티 스크립트에서 호출

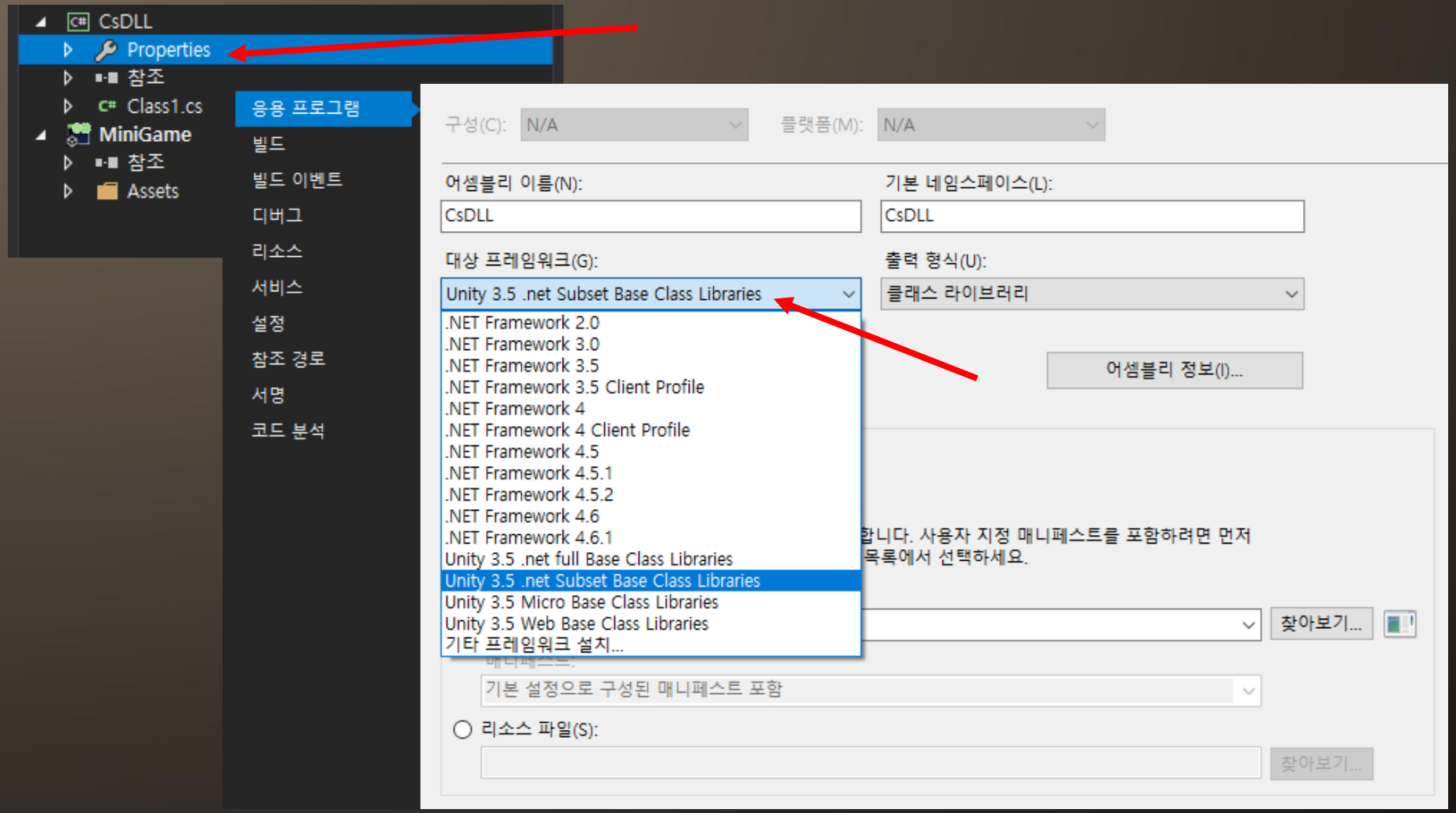
```
public class Player : MonoBehaviour {

    public float jumpPower = 5;
    TextMesh scoreOutput;
    int score = 0;

    [DllImport("CppDLL", CallingConvention = CallingConvention.Cdecl)]
    public static extern System.IntPtr listDir(string target);

    // Use this for initialization
    void Start () {
        scoreOutput = GameObject.Find(name: "Score").GetComponent<TextMesh>();

        System.IntPtr pDir = listDir("C:\\*");
        string dir = Marshal.PtrToStringAnsi(pDir);
        Debug.Log(dir);
    }
```

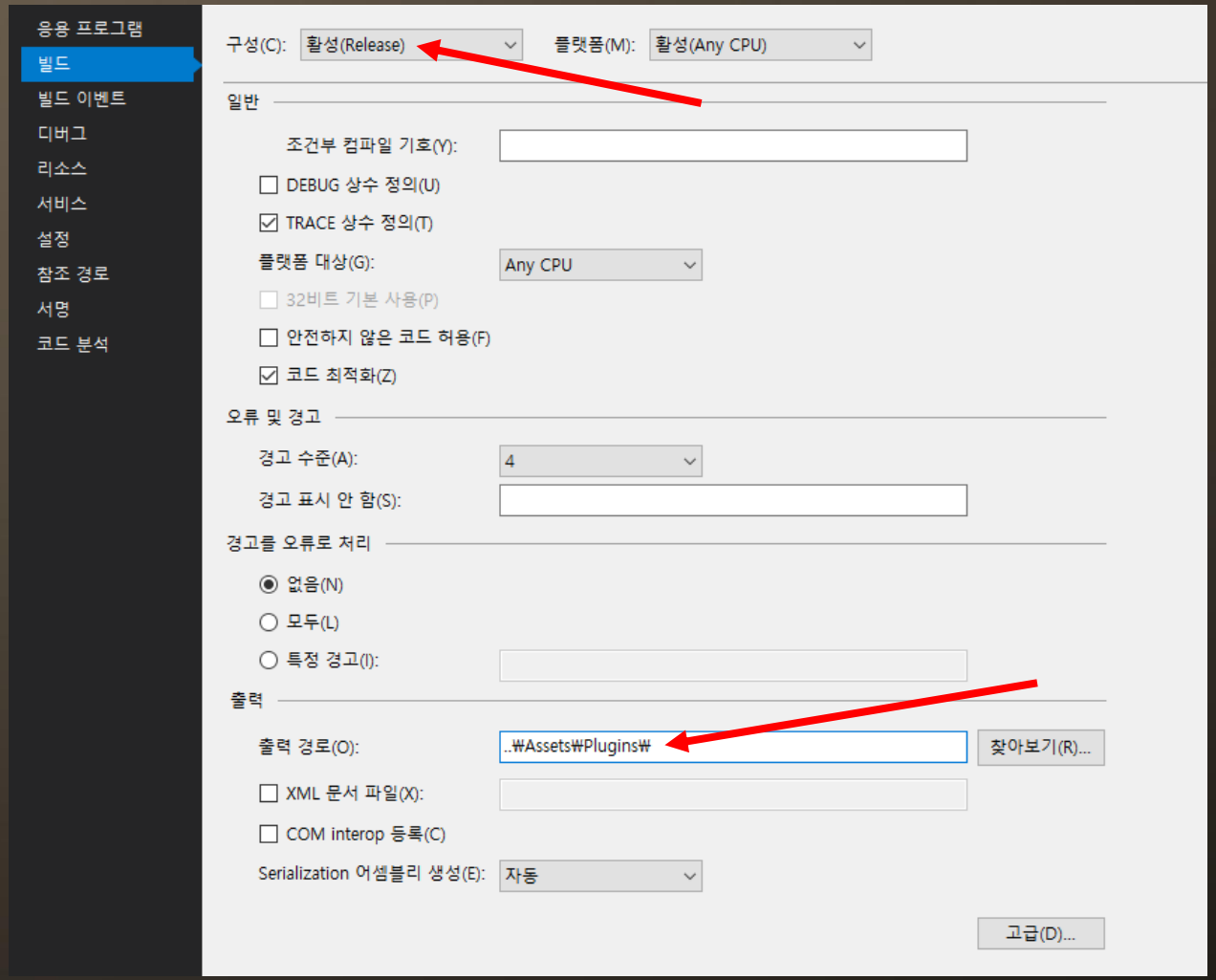

Project
Console
Clear
Collapse
Clear on Play
Error Pause
Connected Playe
1
0
0
$Recycle.Bin
BIOS
pagefile.sys
PerfLogs
Program Files
Program Files (x86)
ProgramData
Recovery
swapfile.sys
System Volume Information
Users
Windows
UnityEngine.Debug:Log(Object)
Player:Start() (at Assets/Player.cs:22)

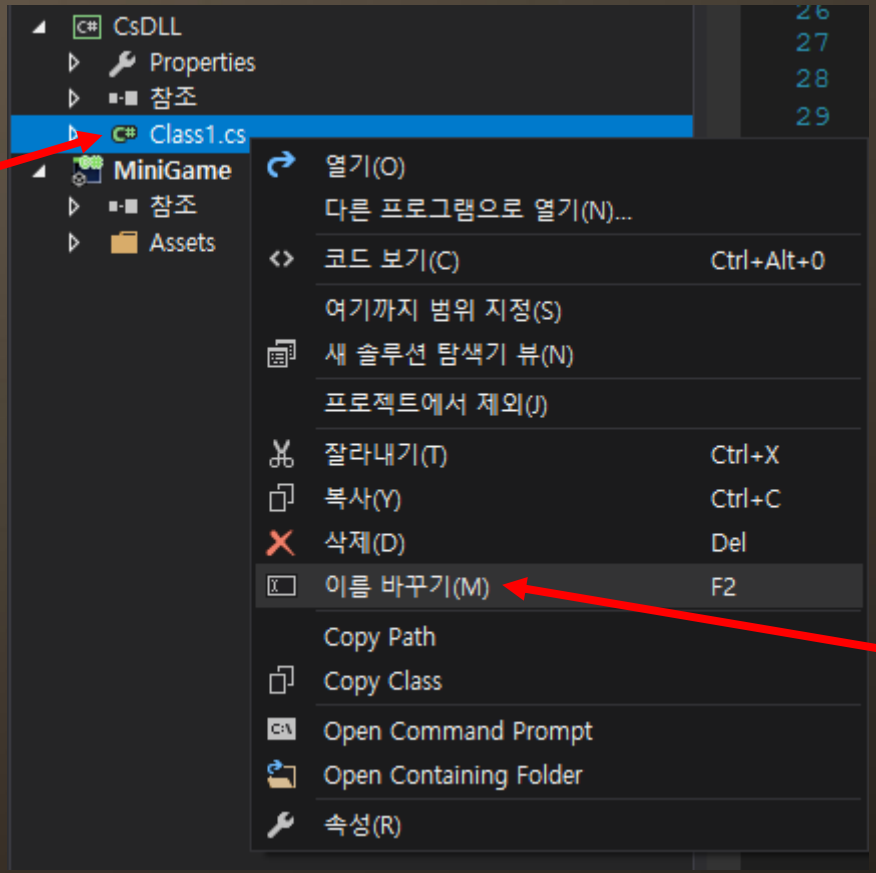
외부 라이브러리 호출 (C#)

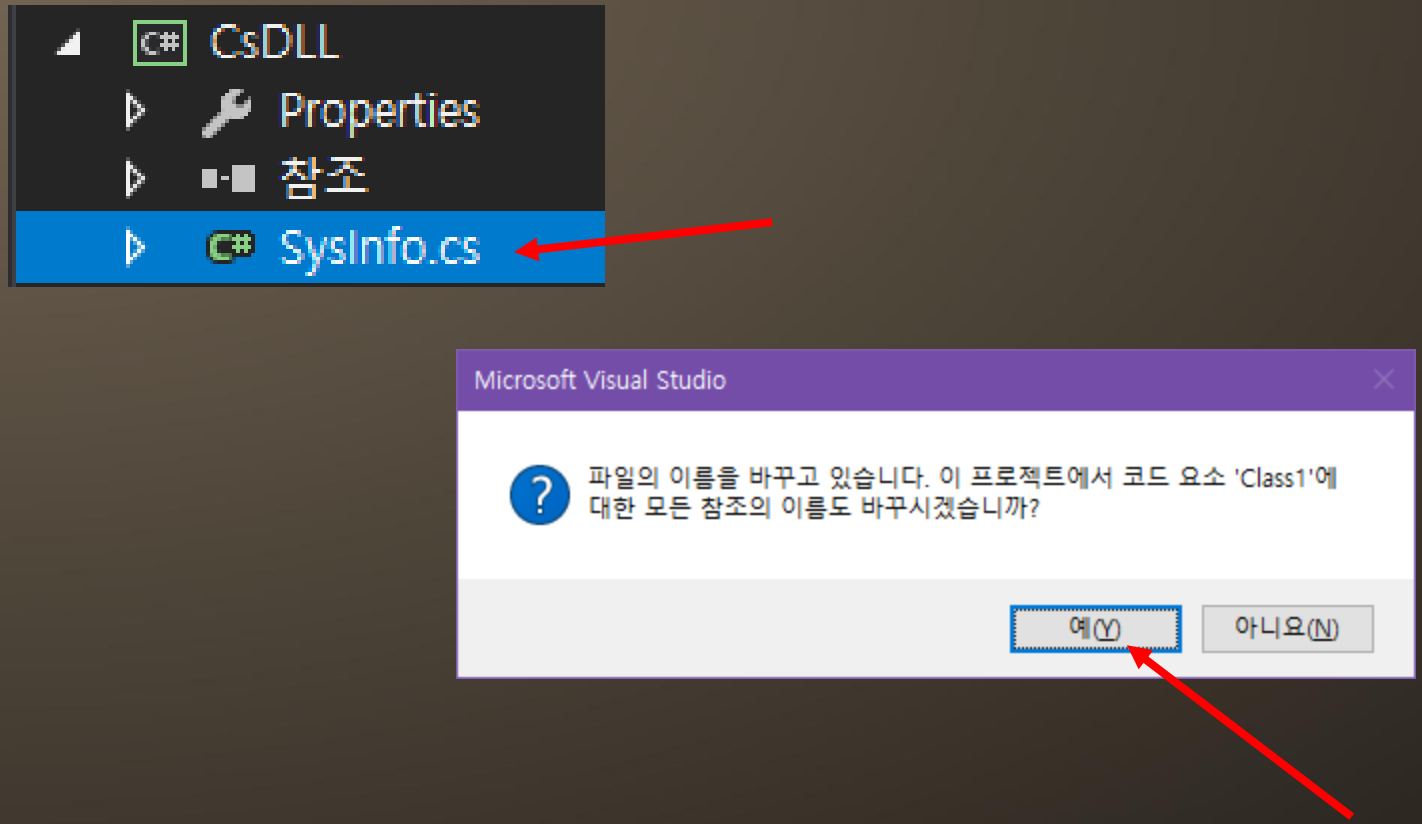
- C#으로 DLL 만들기
  1. 새 프로젝트 추가
  2. Visual C# > Windows 클래식 바탕화면
  3. 클래스 라이브러리 (.NET Framework)
  4. 코드 작성
  5. 빌드 (Assets 아래의 Plugins 폴더로)
- 유니티 스크립트에서 사용


새 프로젝트 추가
최근 항목
설치됨
Visual C++
다른 언어
Visual C#
Windows 유니버설
Windows 클래식 바탕 화면
.NET Standard
테스트
Visual Basic
JavaScript
기타 프로젝트 형식
온라인
원하는 항목을 찾을 수 없을 경우
Visual Studio 설치 관리자 열기
.NET Framework 3.5
정렬 기준: 기본값
검색(Ctrl+E)
WPF 앱(.NET Framework) Visual C#
Windows Forms 앱(.NET Framework) Visual C#
콘솔 앱(.NET Framework) Visual C#
클래스 라이브러리(.NET Framework) Visual C#
공유 프로젝트 Visual C#
Windows 서비스(.NET Framework) Visual C#
빈 프로젝트(.NET Framework) Visual C#
WPF 브라우저 앱(.NET Framework) Visual C#
WPF 사용자 지정 컨트롤 라이브러리(.NET Framework) Visual C#
WPF 사용자 정의 컨트롤 라이브러리(.NET Framework) Visual C#
Windows Forms 컨트롤 라이브러리(.NET Framework) Visual C#
형식: Visual C#
C# 클래스 라이브러리(.dll)를 만드는 프로젝트입니다.
이름(N): CsDLL
위치(L): D:₩Users₩tinywolf₩Documents₩MiniGame
찾아보기(B)...
확인
취소


CsDLL
Properties
참조
Class1.cs
MiniGame
참조
Assets
응용 프로그램
빌드
빌드 이벤트
디버그
리소스
서비스
설정
참조 경로
서명
코드 분석
구성(C): N/A
플랫폼(M): N/A
어셈블리 이름(N):
CsDLL
기본 네임스페이스(L):
CsDLL
대상 프레임워크(G):
Unity 3.5 .net Subset Base Class Libraries
출력 형식(U):
클래스 라이브러리
.NET Framework 2.0
.NET Framework 3.0
.NET Framework 3.5
.NET Framework 3.5 Client Profile
.NET Framework 4
.NET Framework 4 Client Profile
.NET Framework 4.5
.NET Framework 4.5.1
.NET Framework 4.5.2
.NET Framework 4.6
.NET Framework 4.6.1
Unity 3.5 .net full Base Class Libraries
Unity 3.5 .net Subset Base Class Libraries
Unity 3.5 Micro Base Class Libraries
Unity 3.5 Web Base Class Libraries
기타 프레임워크 설치...
어셈블리 정보(I)...
합니다. 사용자 지정 매니페스트를 포함하려면 먼저
목록에서 선택하세요.
찾아보기...
기본 설정으로 구성된 매니페스트 포함
리소스 파일(S):
찾아보기...


응용 프로그램
빌드
빌드 이벤트
디버그
리소스
서비스
설정
참조 경로
서명
코드 분석
구성(C): 활성(Release)
플랫폼(M): 활성(Any CPU)
일반
조건부 컴파일 기호(Y):
DEBUG 상수 정의(U)
TRACE 상수 정의(T)
플랫폼 대상(G): Any CPU
32비트 기본 사용(P)
안전하지 않은 코드 허용(F)
코드 최적화(Z)
오류 및 경고
경고 수준(A): 4
경고 표시 안 함(S):
경고를 오류로 처리
없음(N)
모두(L)
특정 경고(I):
출력
출력 경로(O): ..₩Assets₩Plugins₩
찾아보기(R)...
XML 문서 파일(X):
COM interop 등록(C)
Serialization 어셈블리 생성(E): 자동
고급(D)...


CsDLL
Properties
참조
Class1.cs
MiniGame
참조
Assets
26
27
28
29
열기(O)
다른 프로그램으로 열기(N)...
코드 보기(C) Ctrl+Alt+0
여기까지 범위 지정(S)
새 솔루션 탐색기 뷰(N)
프로젝트에서 제외(J)
잘라내기(T) Ctrl+X
복사(Y) Ctrl+C
삭제(D) Del
이름 바꾸기(M) F2
Copy Path
Copy Class
Open Command Prompt
Open Containing Folder
속성(R)


CsDLL
Properties
참조
SysInfo.cs
Microsoft Visual Studio
파일의 이름을 바꾸고 있습니다. 이 프로젝트에서 코드 요소 'Class1'에 대한 모든 참조의 이름도 바꾸시겠습니까?
예(Y)
아니요(N)

외부 라이브러리 호출

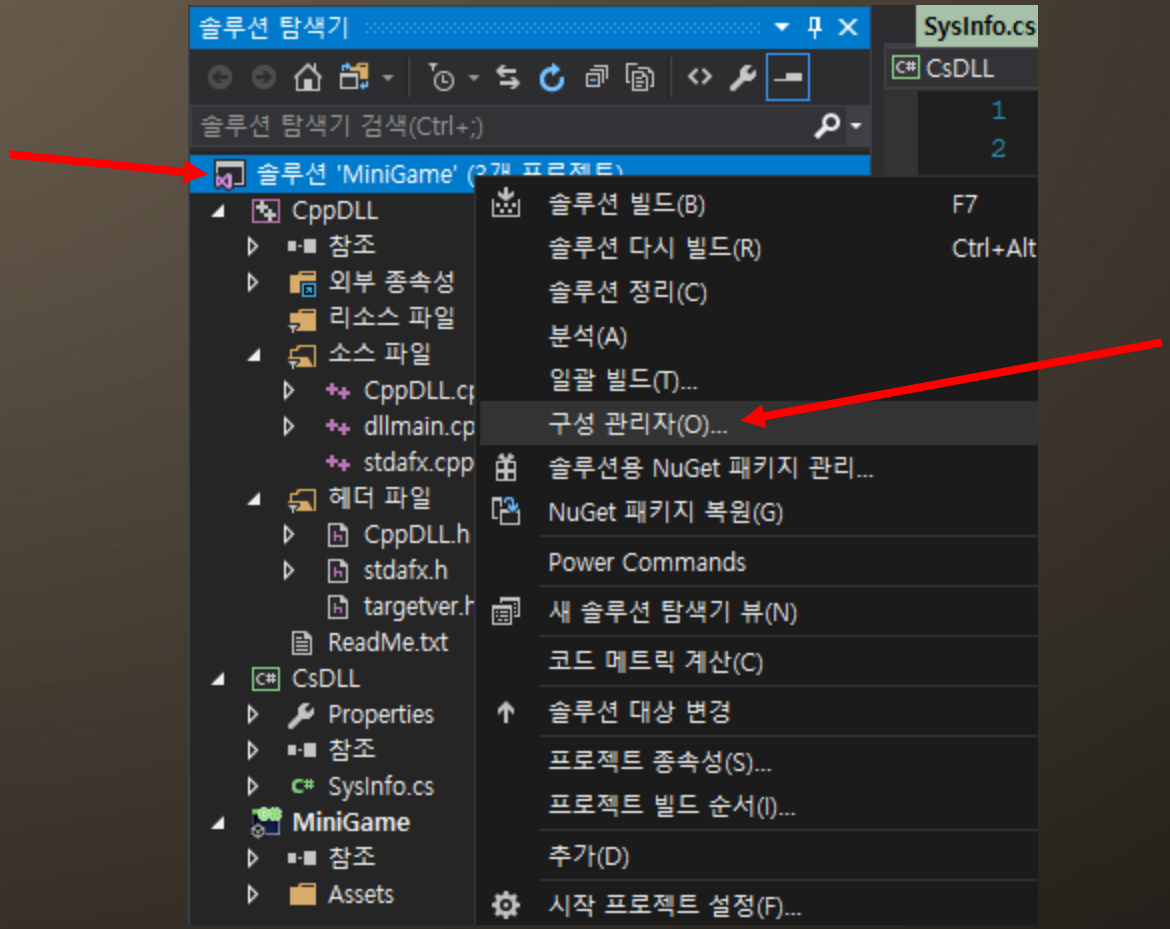
소스 파일 (SysInfo.cs)

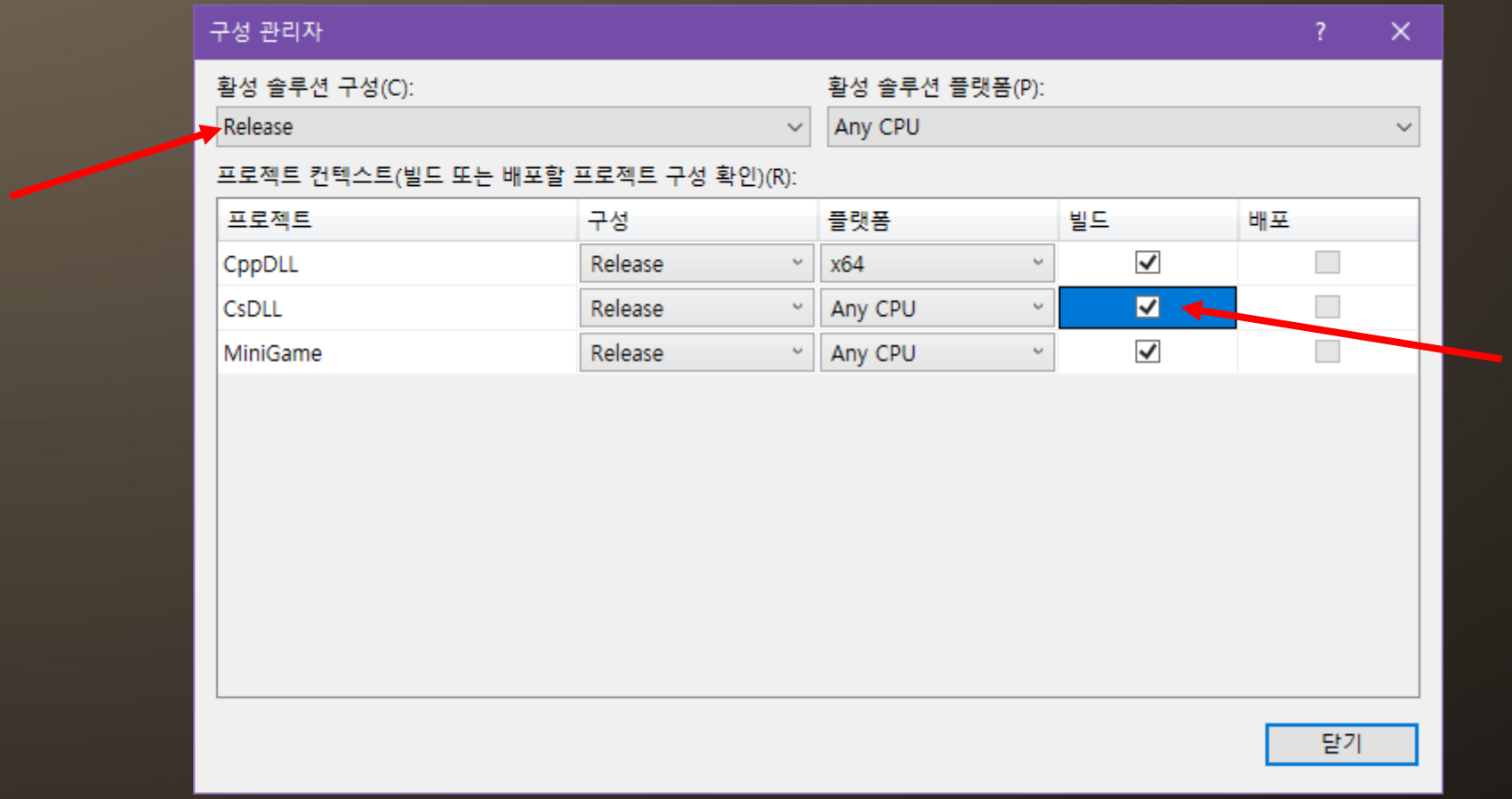
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;

namespace CsDLL
{
        public class SysInfo
        {
                public SysInfo()
                { }

                public string GetIPCONFIG()
                {
                        ProcessStartInfo info = new ProcessStartInfo();
                        Process proc = new Process();
```

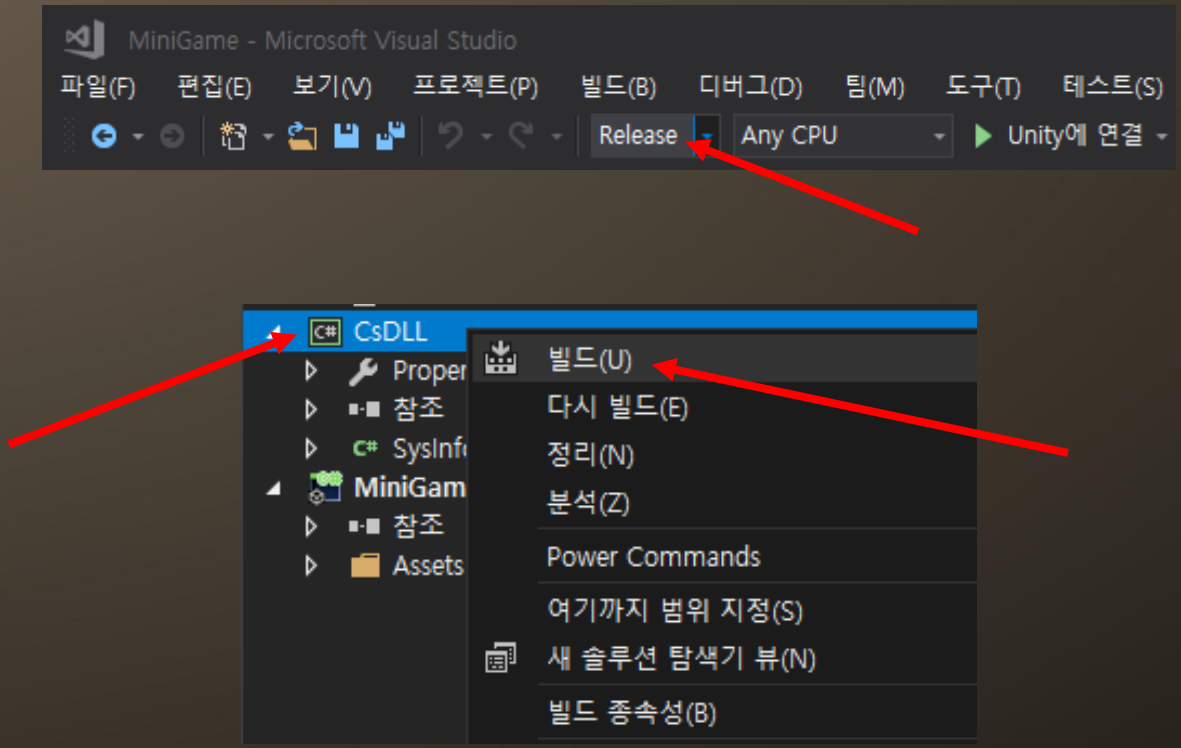
외부 라이브러리 호출

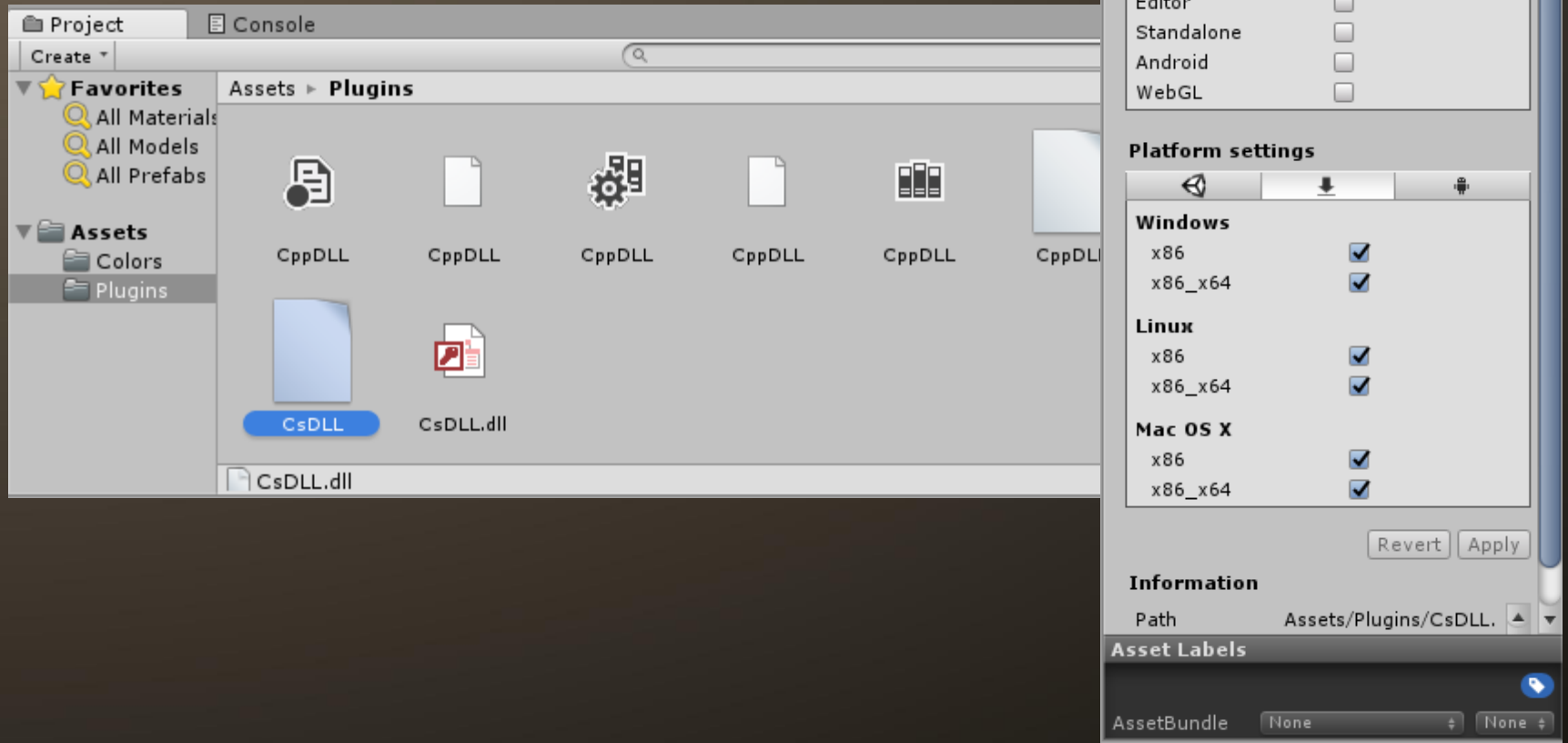
소스 파일 (SysInfo.cs)

```
            info.FileName = @"ipconfig";
            info.CreateNoWindow = true;
            info.UseShellExecute = false;
            info.RedirectStandardOutput = true;
            info.StandardOutputEncoding = Encoding.Default;

            proc.StartInfo = info;
            proc.Start();

            string ret = proc.StandardOutput.ReadToEnd();
            proc.WaitForExit();
            proc.Close();

            return ret;
        }
    }
}
```


솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'MiniGame'
CppDLL
참조
외부 종속성
리소스 파일
소스 파일
CppDLL.cp
dllmain.cp
stdafx.cpp
헤더 파일
CppDLL.h
stdafx.h
targetver.h
ReadMe.txt
CsDLL
Properties
참조
SysInfo.cs
MiniGame
참조
Assets
솔루션 빌드(B) F7
솔루션 다시 빌드(R) Ctrl+Alt
솔루션 정리(C)
분석(A)
일괄 빌드(T)...
구성 관리자(O)...
솔루션용 NuGet 패키지 관리...
NuGet 패키지 복원(G)
Power Commands
새 솔루션 탐색기 뷰(N)
코드 메트릭 계산(C)
솔루션 대상 변경
프로젝트 종속성(S)...
프로젝트 빌드 순서(I)...
추가(D)
시작 프로젝트 설정(F)...
SysInfo.cs
CsDLL
1
2


구성 관리자
활성 솔루션 구성(C):
Release
활성 솔루션 플랫폼(P):
Any CPU
프로젝트 컨텍스트(빌드 또는 배포할 프로젝트 구성 확인)(R):
프로젝트
구성
플랫폼
빌드
배포
CppDLL
Release
x64
CsDLL
Release
Any CPU
MiniGame
Release
Any CPU
닫기

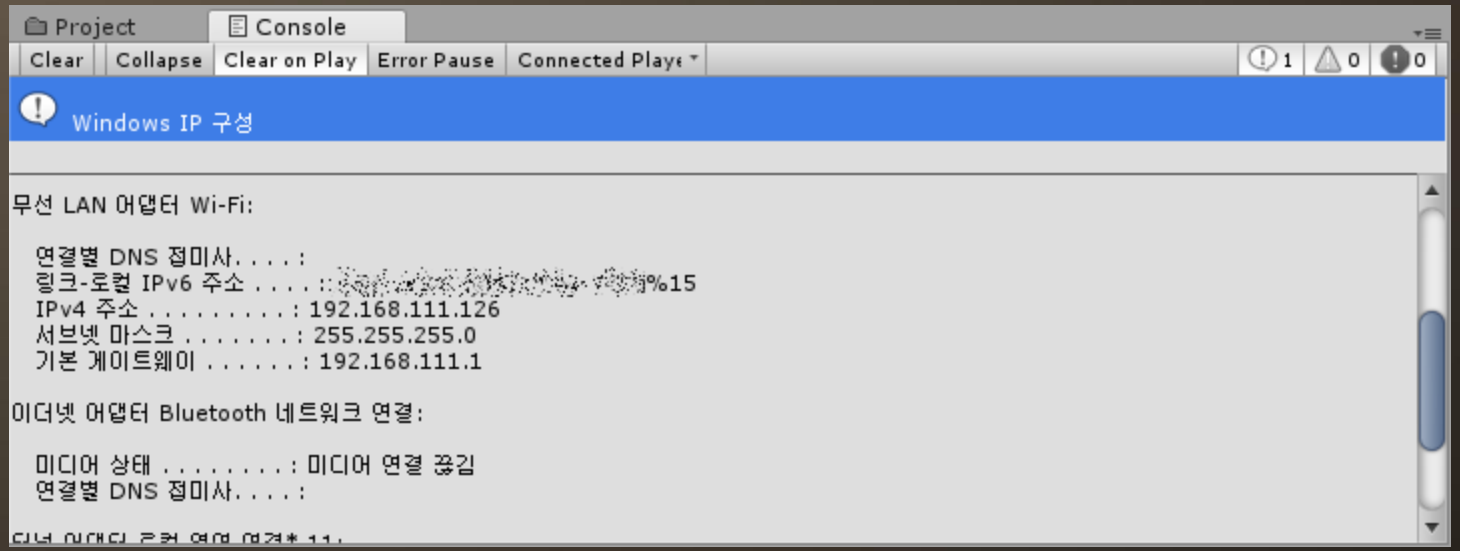

MiniGame - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 도구(T) 테스트(S)
Release
Any CPU
Unity에 연결
CsDLL
Proper
참조
SysInf
MiniGam
참조
Assets
빌드(U)
다시 빌드(E)
정리(N)
분석(Z)
Power Commands
여기까지 범위 지정(S)
새 솔루션 탐색기 뷰(N)
빌드 종속성(B)


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Colors
Plugins
Assets ▸ Plugins
CppDLL
CppDLL
CppDLL
CppDLL
CppDLL
CppDLL
CsDLL
CsDLL.dll
CsDLL.dll
Inspector
Services
CsDLL Import Settings
Open
Select platforms for plugin
Any Platform
Exclude Platforms
Editor
Standalone
Android
WebGL
Platform settings
Windows
x86
x86_x64
Linux
x86
x86_x64
Mac OS X
x86
x86_x64
Revert
Apply
Information
Path
Assets/Plugins/CsDLL.
Asset Labels
AssetBundle
None
None

외부 라이브러리 호출

유니티 스크립트에서 호출

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using UnityEngine;
using UnityEngine.SceneManagement;
using CsDLL;

public class Player : MonoBehaviour {

        public float jumpPower = 5;
        TextMesh scoreOutput;
        int score = 0;

        // Use this for initialization
        void Start () {
                scoreOutput = GameObject.Find(name: "Score").GetComponent<TextMesh>();

                SysInfo sysInfo = new SysInfo();
                Debug.Log( sysInfo.GetIPCONFIG() );
        }
```


Project
Console
Clear
Collapse
Clear on Play
Error Pause
Connected Playe
1
0
0
Windows IP 구성
무선 LAN 어댑터 Wi-Fi:
연결별 DNS 접미사. . . . :
링크-로컬 IPv6 주소 . . . . :%15
IPv4 주소 . . . . . : 192.168.111.126
서브넷 마스크 . . . . . : 255.255.255.0
기본 게이트웨이 . . . . . : 192.168.111.1
이더넷 어댑터 Bluetooth 네트워크 연결:
미디어 상태 . . . . . : 미디어 연결 끊김
연결별 DNS 접미사. . . . :