

# R-eve

지도교수 : 정지영

<https://url.kr/fceigz>

# RECAP

C#

```
delegate void CalcDele(int i, int j);
```

```
static void Main(string[] args)    {  
    bool check1 = int.TryParse(Console.ReadLine(), out int i);  
    ConsoleKeyInfo k = Console.ReadKey();  
    bool check2 = int.TryParse(Console.ReadLine(), out int j);  
    switch (k.Key)    {  
        case ConsoleKey.Add:  
            Calc(i, j, Add);  
            break;  
        case ConsoleKey.Subtract:  
            Calc(i, j, Sub);  
            break;  
        case ConsoleKey.Multiply:  
            Calc(i, j, Mul);  
            break;  
        default:  
            Console.WriteLine("Error!");  
            break;  
    }  
}
```

메서드를 매개변수로 전달

```
private static void Calc(int i, int j, CalcDele operate)
```

- Swap과 SwapRef apjem 호출시 메모리의 변화를 그려보시오.

```
static void Main(string[] args)
{
    int a = 100;
    int b = 200;

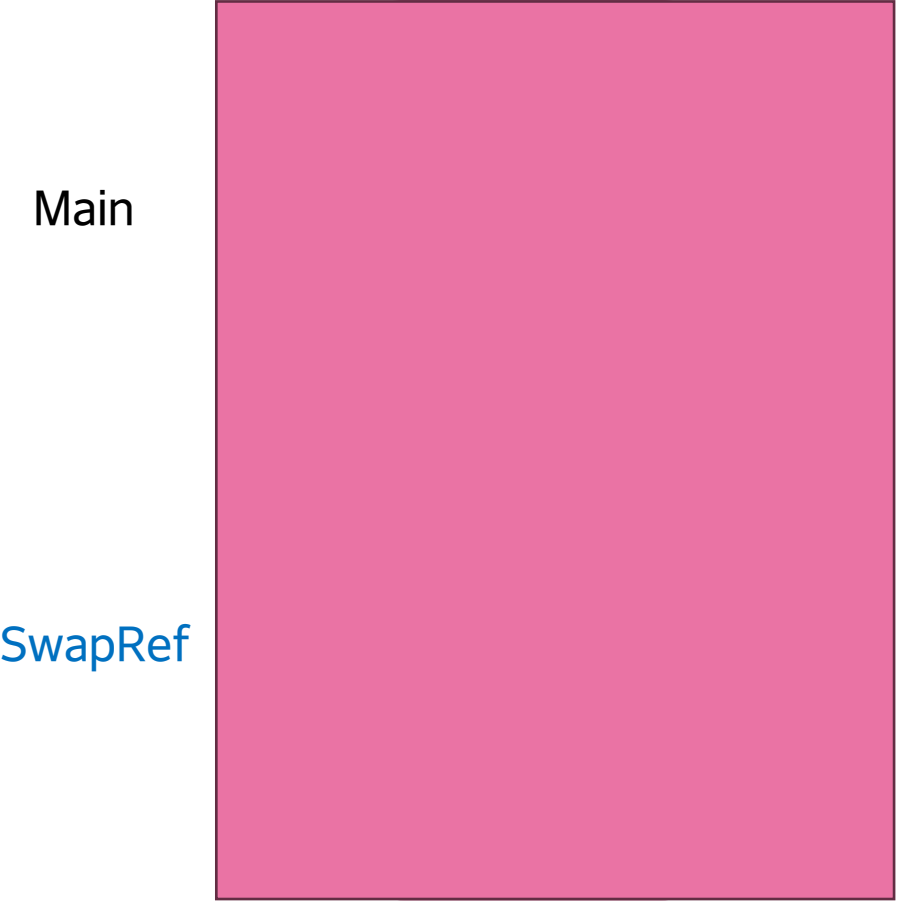
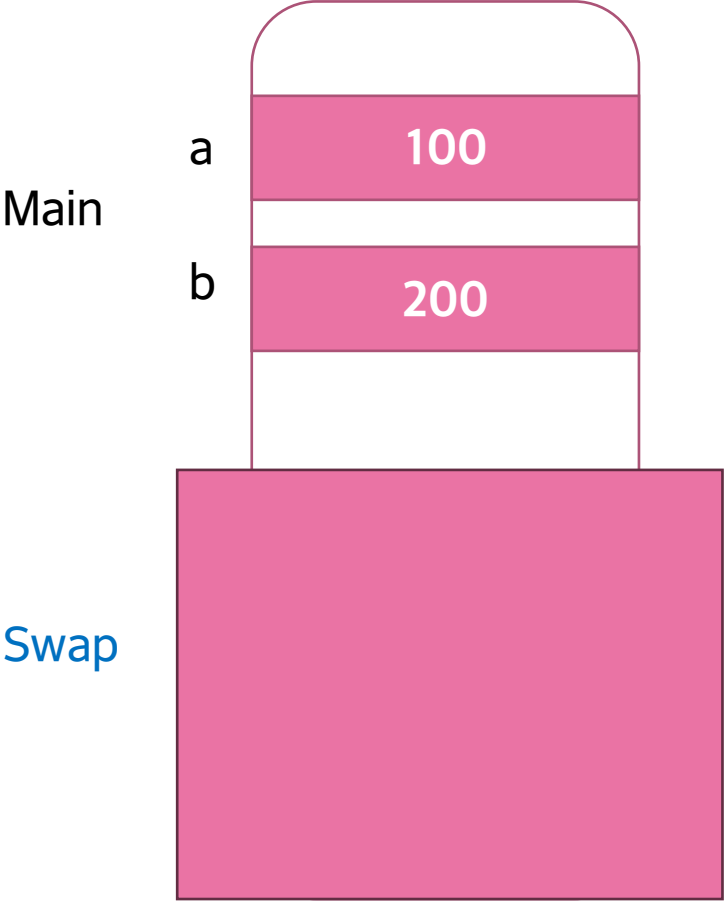
    Swap(a, b);
    Console.WriteLine(a); // 100
    Console.WriteLine(b); // 200

    SwapRef(ref a, ref b);
    Console.WriteLine(a); // 200
    Console.WriteLine(b); // 100
}
```

```
private static void Swap(int ap, int bp)
{
    int temp = ap;
    ap = bp;
    bp = temp;
}

private static void SwapRef(ref int ap, ref int bp)
{
    int temp = ap;
    ap = bp;
    bp = temp;
}
```

# Swap vs SwapRef



# Swap List

```
class Program
{
    static void Main(string[] args)
    {
        int[] a = { 3, 5 };
        SwapArray(ref a[0], ref a[1]);
        Console.WriteLine($"{a[0]} {a[1]}"); //5, 3
    }

    private static void SwapArray(ref int a, ref int b)
    {
        int t = a;
        a = b;
        b = t;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        List<int> a = new List<int>() { 3, 5 };
        Swap(a);
        Console.WriteLine($"{a[0]} {a[1]}");
    }

    private static void Swap(List<int> a)
    {
        int t = a[0];
        a[0] = a[1];
        a[1] = t;
    }
}
```

## ■ 깊은 복사(Deep Copy) vs 얕은 복사(Shallow Copy)

```
class Test
{
    public int value;

    public Test DeepCopy()
    {
        Test newTest = new Test();
        newTest.value = this.value;

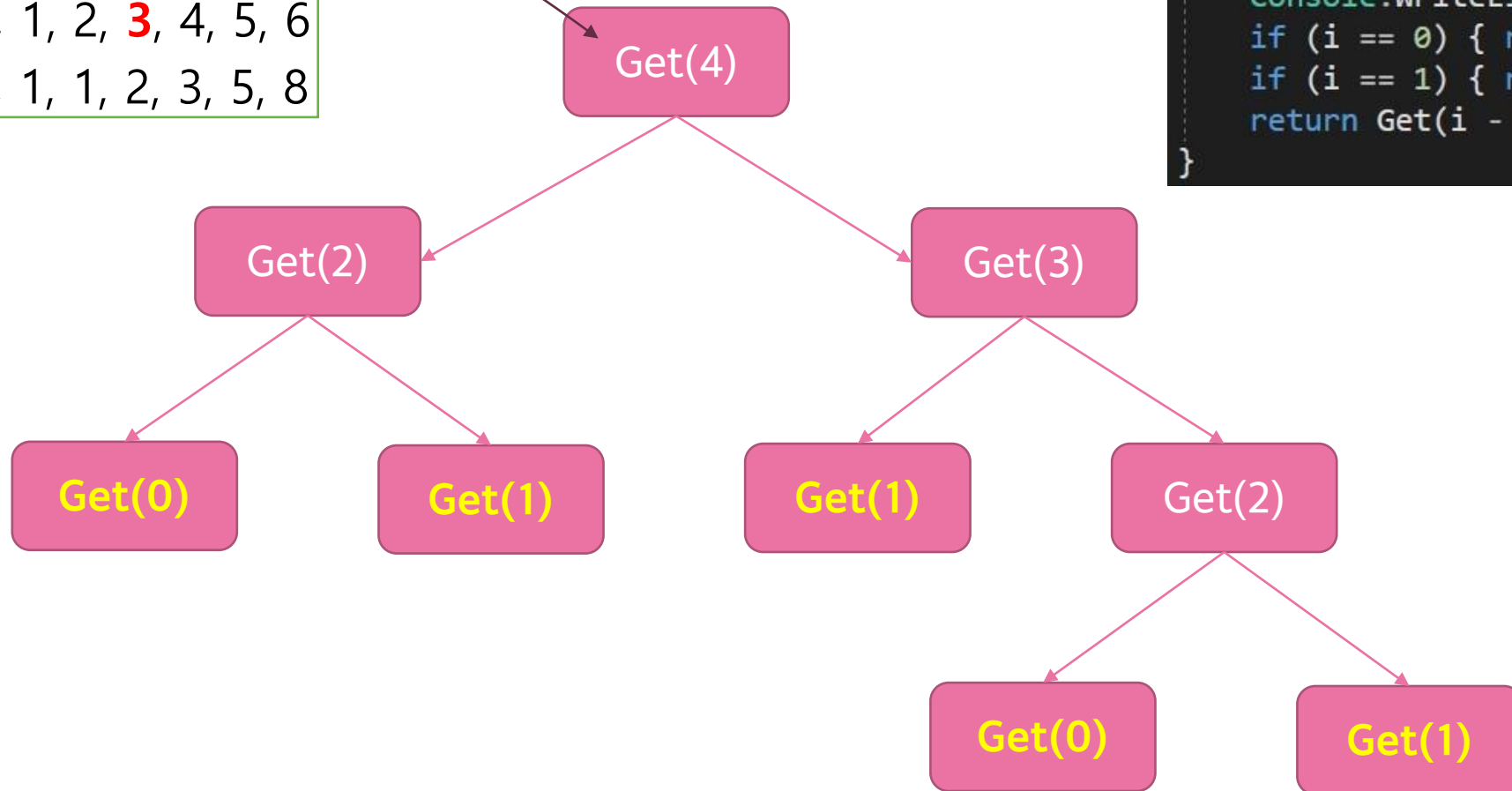
        return newTest;
    }
}
```

# 재귀 : 피보나치 수열 Get(4)의 재귀과정

## 피보나치 수

Index : 0, 1, 2, **3**, 4, 5, 6

Value : 0, 1, 1, 2, 3, 5, 8



```
public static long Get(int i)
{
    Console.WriteLine(i);
    if (i == 0) { return 0; }
    if (i == 1) { return 1; }
    return Get(i - 2) + Get(i - 1);
}
```



## Section 10 함께하는 응용예제(4)

- 응용예제 6-2 메모화 /6장/Memorize

- 메모화 : 한 번 계산했던 값을 저장해두는 것

코드 6-37 메모화

```
class Fibonacci
{
    private static Dictionary<int, long> memo = new Dictionary<int, long>();
    public static long Get(int i)
    {
        // 기본 값
        if (i < 0) { return 0; }
        if (i == 1) { return 1; }

        // 이미 계산했던 값인지 확인
        if (memo.ContainsKey(i))
        {
            return memo[i];
        }
        else
        {
            계산한 피보나치 수를 저장하는 Dictionary 객체를 만듭니다.
        }
    }
}
```

## Section 03 out 키워드(1)

- 값을 여러 개 반환하고자 할 때 사용, 대표적인 메서드: TryParse ( )
- TryParse ( ) 메서드는 숫자로 바꿀 수 있는 문자열을 매개변수로 넣으면 true 반환
- 바꿀 수 없는 문자열을 매개변수로 넣으면 false 반환
- 문자열을 숫자로 변환 결과는 반환 않고 매개변수 out int result 에 넣은 변수로 반환

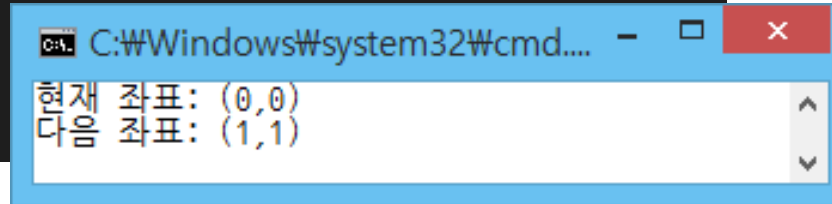
```
while (true) {  
    bool flag = int.TryParse(Console.ReadLine(), out int result);  
    if (flag) {  
        break;  
    }  
    else {  
        Console.WriteLine("re-enter");  
    }  
}
```

Int 입력이 들어올때까지 반복

## Section 03 out 키워드(3)

- out 키워드를 사용하는 메서드 생성

```
static void NextPosition(int x, int y, int vx, int vy, out int rx, out int ry)
{
    // 다음 위치 = 현재 위치 + 현재 속도
    rx = x + vx;
    ry = y + vy;
}
```



C:\Windows\system32\cmd...  
현재 좌표: (0,0)  
다음 좌표: (1,1)

- Convert vs. Parse vs. TryParse

- \* Convert : 기본 데이터 형식을 다른 기본 데이터 형식으로 변환.
- \* Parse : 문자열 표현을 해당하는 형 으로 변환.
- \* TryParse : 문자열 표현을 해당하는 형 으로 변환. 반환 값은 변환의 성공 여부를 나타낸다.

Convert.ToInt32(v)

Convert.ToString(v)

Int.Parse(v)

Float.Parse(v)

Int.TryParse(v, out int w)

Float.TryParse(v, out float w)

## Section 04 구조체(1)

- 구조체를 만드는 기본 방법은 클래스와 유사하며 간단한 객체 만들 때 사용
- 구조체는 상속과 인터페이스 구현이 불가능 함, 클래스보다 안정성 높음
- C#의 기본 자료형은 모두 구조체로 정의
- 클래스(reference type)와 구조체(value type)는 복사 형식이 다름

C#은 기본적으로 모든 클래스의 인스턴스 변수는 포인터이고,  
모든 구조체의 인스턴스 변수는 값 형식

```
struct Point  
{  
    public int x;  
    public int y;  
}
```

### ■ 구조체의 생성자

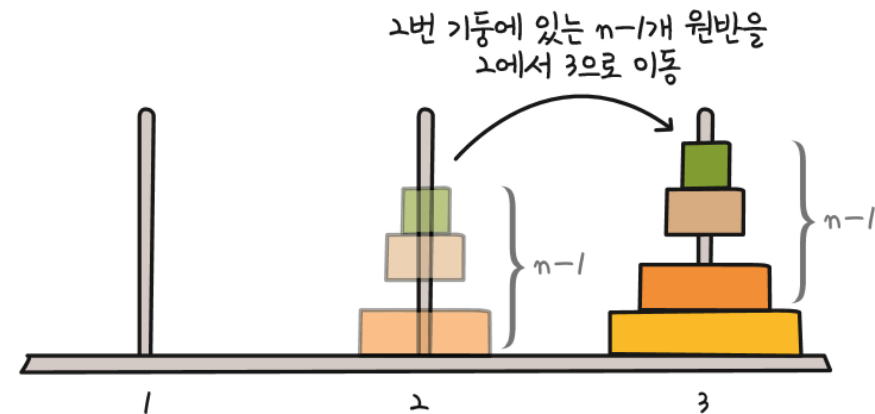
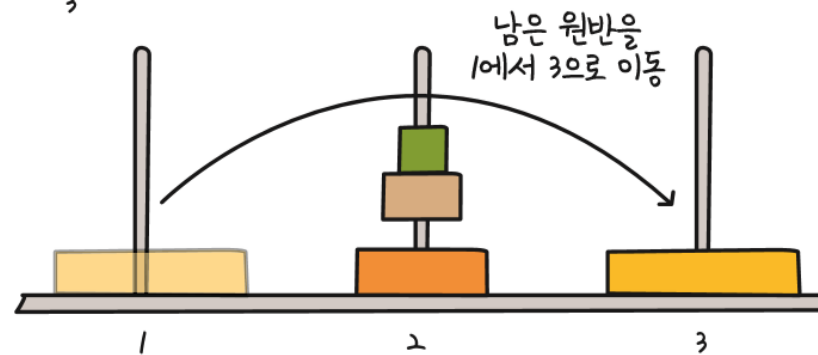
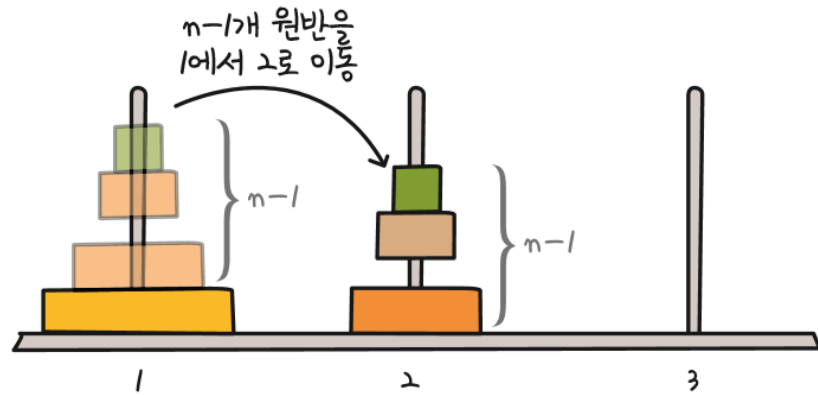
- 매개변수 없는 생성자 선언 불가
- 매개변수 없는 생성자가 자동 정의되어 구조체는 매개변수 없는 생성자 만들 수 없음
- 매개변수 있는 생성자 만들어도, 매개변수 없는 생성자 사용 가능
- 내부 변수는 자동적으로 해당 자료형의 기본 값으로 초기화

The background is a complex, abstract geometric pattern composed of numerous triangles of various sizes and colors. The colors include shades of yellow, orange, red, pink, purple, blue, and green, creating a vibrant, mosaic-like effect. The triangles are arranged in a way that they overlap and interlock, forming a continuous, non-repeating pattern.

# Hanoi Tower

Hanoi

# Hanoi

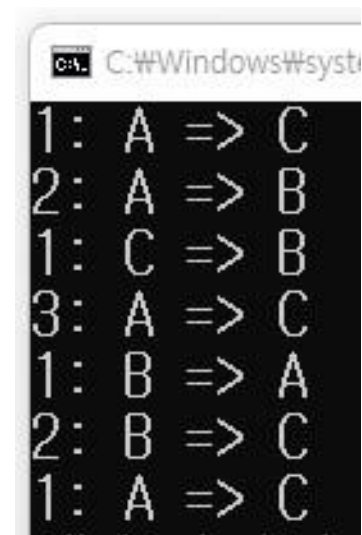


# C# 코드

- 재귀 호출이 정상 작동하려면 '종료 조건' 필요

```
class Program
{
    public static void Main(String[] args)
    {
        Hanoi hanoi = new Hanoi();
        hanoi.hanoiMove(3, "A", "B", "C");
    }
}

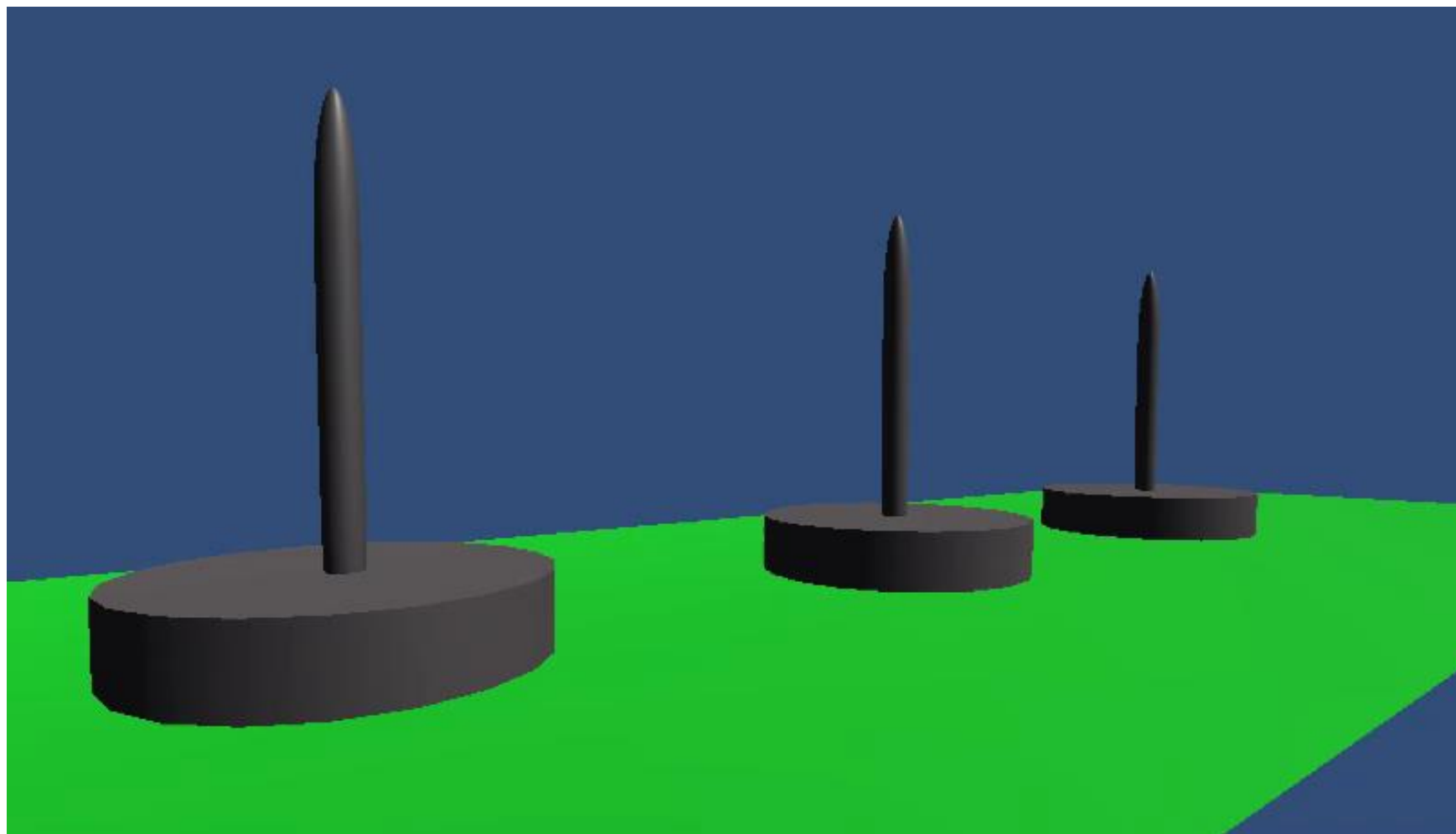
public class Hanoi
{
    public void hanoiMove(int n, string src, string temp, string dest)
    {
        if (n == 1)
        {
            Console.WriteLine(n + ": " + src + " => " + dest);
            return;
        }
        hanoiMove(n - 1, src, dest, temp);
        Console.WriteLine(n + ": " + src + " => " + dest);
        hanoiMove(n - 1, temp, src, dest);
    }
}
```



```
C:\Windows\system32\cmd.exe
1: A => C
2: A => B
1: C => B
3: A => C
1: B => A
2: B => C
1: A => C
```



# 실행결과



# C# tuple

Tuple, ValueTuple은 C# 7.0에 처음 도입

**클래스 튜플 타입**(전달할 때 복사가 발생하지 않고, 참조를 전달)

```
Tuple<int, float> tuple = new Tuple<int, float>(10, 20f); // 8개까지 지정가능
```

**구조체 튜플 타입**(전달할 때마다 복사 발생)

```
ValueTuple<int, float, string> tuple = (1, 0.5f, "ok");
```

```
(int, float, string) tuple = (1, 0.5f, " ok ");
```

```
var tuple = (1, 0.5f, " ok ");
```

tuple에 대한 각각의 필드는 `.item1`, `.item2`, ...로 참조가능하고 이름을 줄 수 도 있음

```
- (int id, string name) tuple = (1, " ok ");
```

# C# tuple 활용

// 충돌점의 위치 및 방향 벡터를 저장하는 tuple의 리스트

```
public List<(Vector3 pt,Vector3 dir)> pointDir = new List<(Vector3,Vector3)>();
```

```
pointDir.Clear(); // tuple 비우기
```

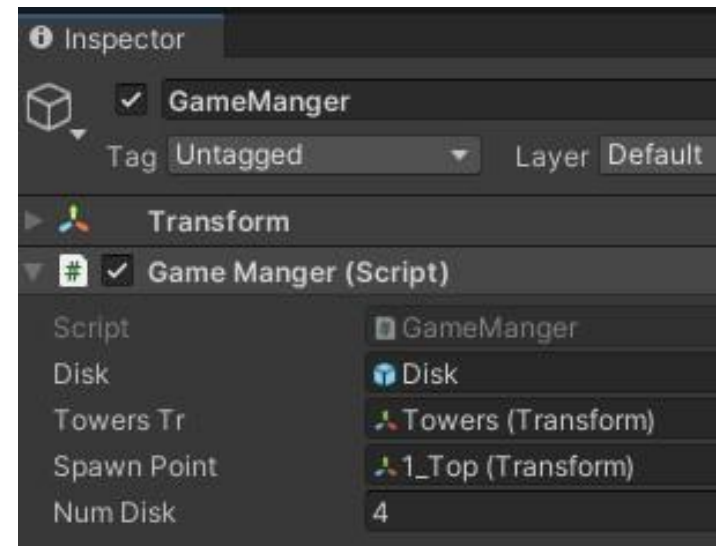
```
pointDir.Add((transform.position, playerTr.position)); // tuple에 추가, ()로 묶음
```

```
CheckCollision(pointDir[i].pt, pointDir[i].dir); // i번째 tuple의 각 item 접근
```

# Unity 코드

```
public GameObject disk;
public Transform towersTr;
public Transform spawnPoint;
List<(int start,int end)> list = new List<(int, int)>(); // 원반 이동 과정 저장
public int numDisk;
int moveCount = 0;
void Start() {
    StartCoroutine(Spawn());
}

IEnumerator Spawn() {
    // 블럭을 Disk수 만큼 만들어 spawnPoint에 배치
    for (int i = 0; i < numDisk; i++)
    {
        GameObject instance = Instantiate(disk, spawnPoint);
        instance.name = (i + 1).ToString();
        instance.transform.parent = spawnPoint;
        instance.transform.localScale = new Vector3(3f - (i * 0.3f), 0.1f, 3.0f - (i * 0.3f));
        yield return new WaitForSeconds(0.5f);
    }
}
```

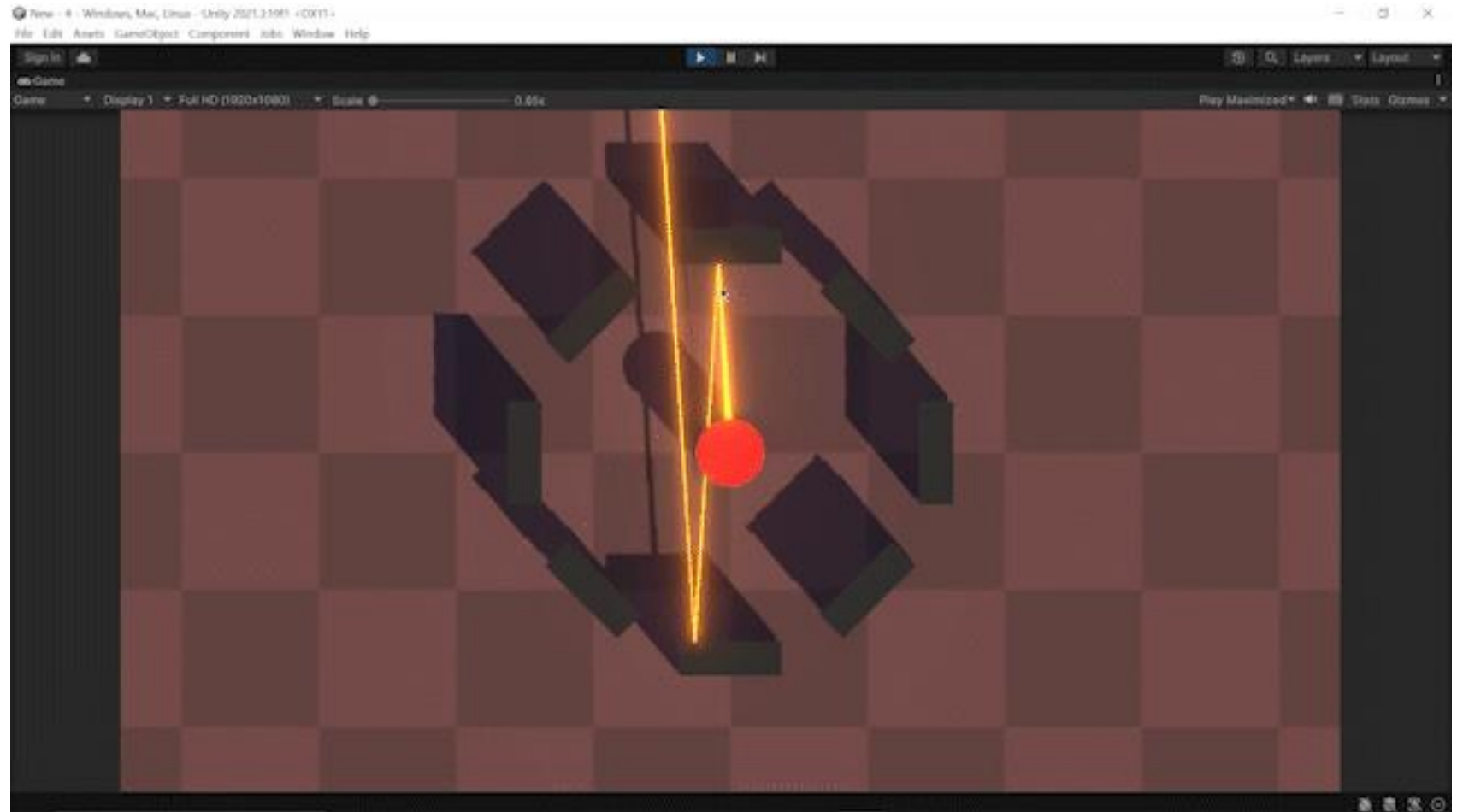
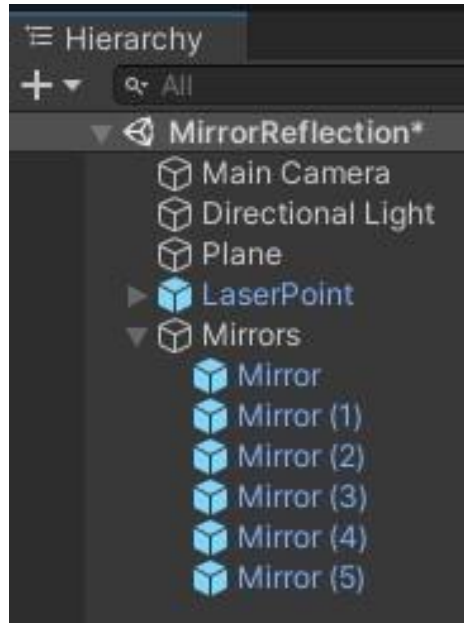


The background is a complex, abstract geometric pattern composed of numerous overlapping triangles of various sizes and colors. The colors include shades of yellow, orange, red, pink, purple, blue, and green, creating a vibrant, mosaic-like effect. The triangles are arranged in a way that creates a sense of depth and movement.

# Mirror Reflection

Mirror Reflection

# 실행 결과

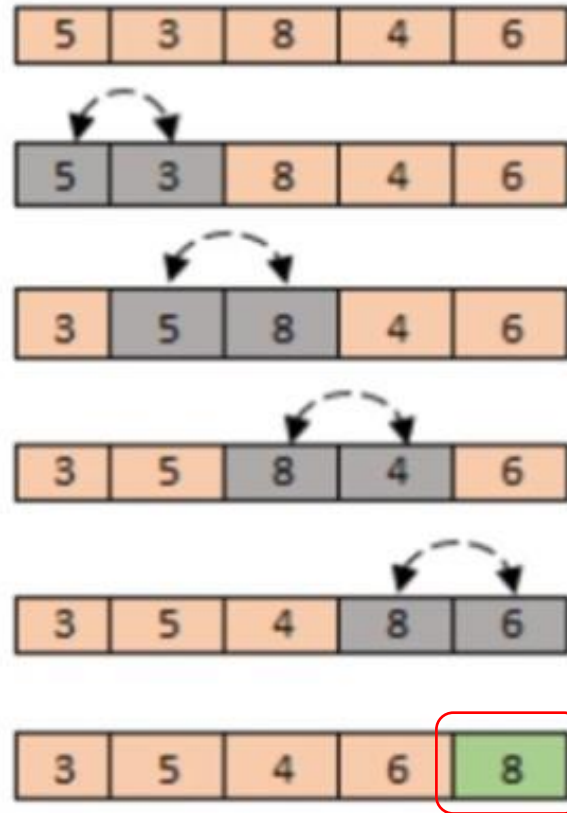




The background is a complex, abstract geometric pattern composed of numerous triangles of various sizes and colors, including shades of yellow, orange, red, pink, purple, blue, and green. The triangles are arranged in a way that creates a sense of depth and movement. A white rectangular box is positioned in the lower center of the image, containing the text '정렬 (Sort)'. Below this box is a solid pink horizontal bar.

## 정렬 (Sort)

# 버블정렬 : 시간복잡도 = $O(n^2)$



숫자 2개를 비교하여  
큰 수를 오른쪽으로 보내기

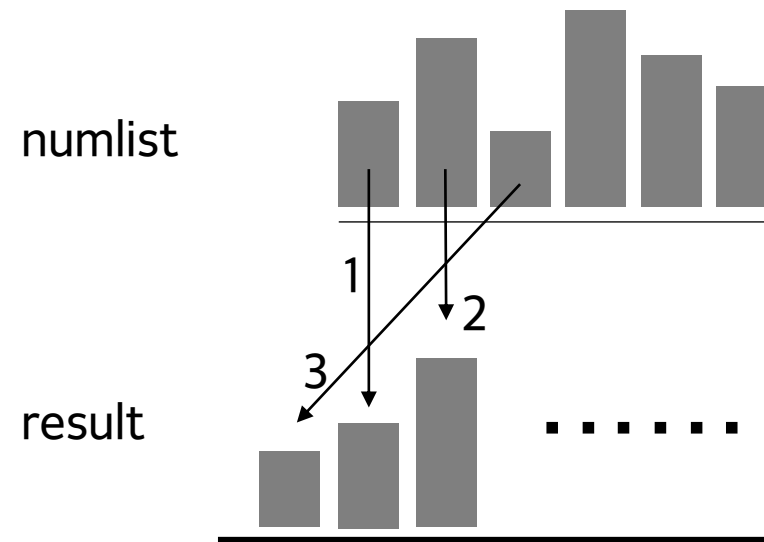
cf.  
`Array.Sort(array);`

완료



# 삽입 정렬

- 앞에서부터 하나씩 꺼내서 순서에 맞는곳에 삽입
- 이미 정렬이 끝난 리스트의 경우는  $O(n)$



# 정렬 알고리즘 정리

■ <https://jinhyy.tistory.com/9>

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
✓ 삽입정렬	$n$	$n^2$	$n^2$	7.438
✓ 선택정렬	$n^2$	$n^2$	$n^2$	10.842
✓ 버블정렬	$n^2$	$n^2$	$n^2$	22.894
셀 정렬	$n$	$n^{1.5}$	$n^2$	0.056
✓ 퀵 정렬	$n \log_2 n$	$n \log_2 n$	$n^2$	0.014
✓ 힙 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.034
✓ 병합정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.026

<https://kangworld.tistory.com/41>


- Python, C, Java 등 대부분 언어의 정렬은 퀵정렬과 병합정렬의 하이브리드를 사용

■ <https://velog.io/@wishtree/퀵정렬-병합정렬-힛정렬-정의와-비교분석>


	퀵정렬	병합정렬	힛정렬
특징	불안정, 제자리, 피벗 있는 분할정복	안정, 제자리, 원본크기 메모리 필요	불안정, 제자리, 트리
평균	$n \log n$	$n \log n$	$n \log n$
최악	$n^2$	$n \log n$	$n \log n$
최선	$n \log n$	$n \log n$	$n \log n$
공간복잡도	$n$	$n$	$n$

■ Visualization and Comparison of Sorting Algorithms

- <https://www.youtube.com/watch?v=ZZuD6iUe3Pc>



# C# 과제



- 0. 영상강의 시청 (소콘과 1기 졸업 선배 조언) (선택과제)
- 1. 랜덤한 정수 10개(0~99)를 생성하여 배열에 저장하고 이 배열에 들어있는 숫자를 크기 순으로 버블정렬(필수과제) 및 삽입정렬(선택과제) 프로그램을 작성하시오.(C#이 제공하는 Sort함수 이용하지 않는 버전)
- 2. 문자열이 거꾸로 읽어도 같은 문장인지 판단하여 회문이면 **True**, 아니면 **False**를 결과로 알려주는 알고리즘을 프로그래밍 하시오. (필수과제)

회문의 예(한글)	회문의 예(영어)
역삼역	mom
기러기	wow
일요일	noon
사진사	level
복불복	radar
다가가다	kayak
기특한 특기	racecar
다했나? 했다!	God's dog
다시 합창 합시다	Madam, I'm Adam.

