

The background of the slide is composed of numerous overlapping triangles in various colors including yellow, orange, red, pink, purple, blue, and green, creating a mosaic-like effect. A large white rectangular area is centered on the slide, serving as a backdrop for the title and instructor information.

R-eve

지도교수 : 정지영

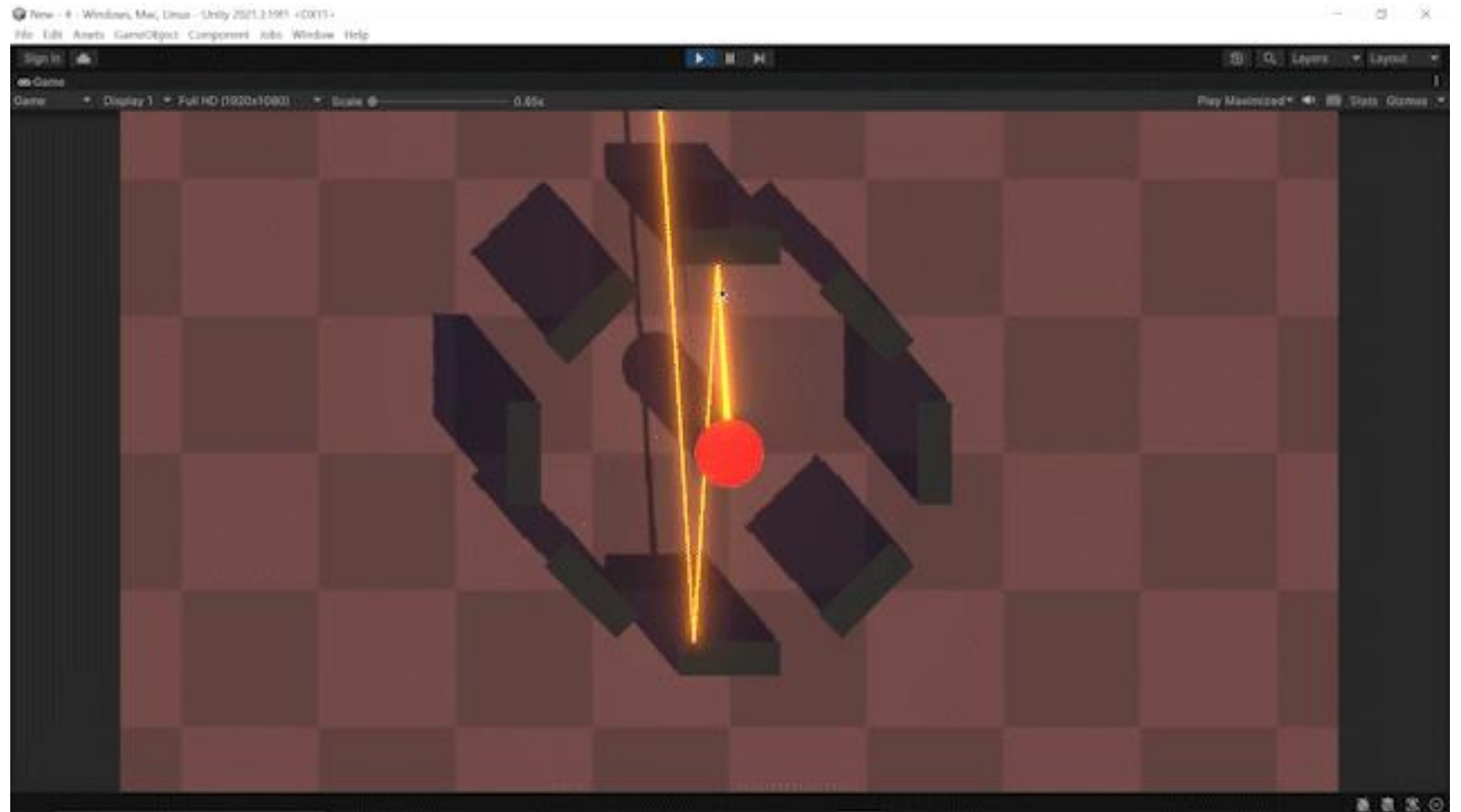
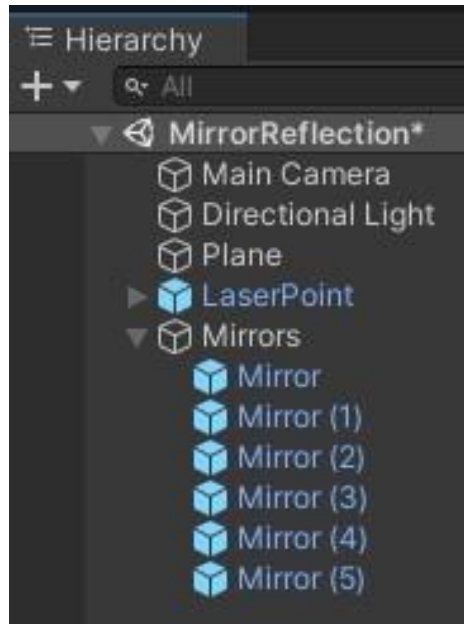
<https://url.kr/fceigz>

The background is a complex, abstract geometric pattern composed of numerous overlapping triangles of various sizes. The colors are vibrant and varied, 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, with some triangles appearing to be in the foreground and others receding into the background. The overall effect is a dynamic and colorful mosaic.

Mirror Reflection

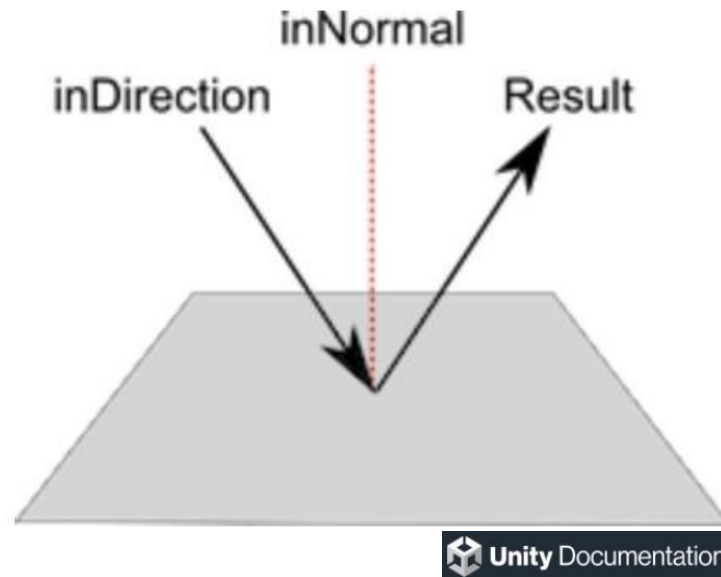
Mirror Reflection

실행 결과

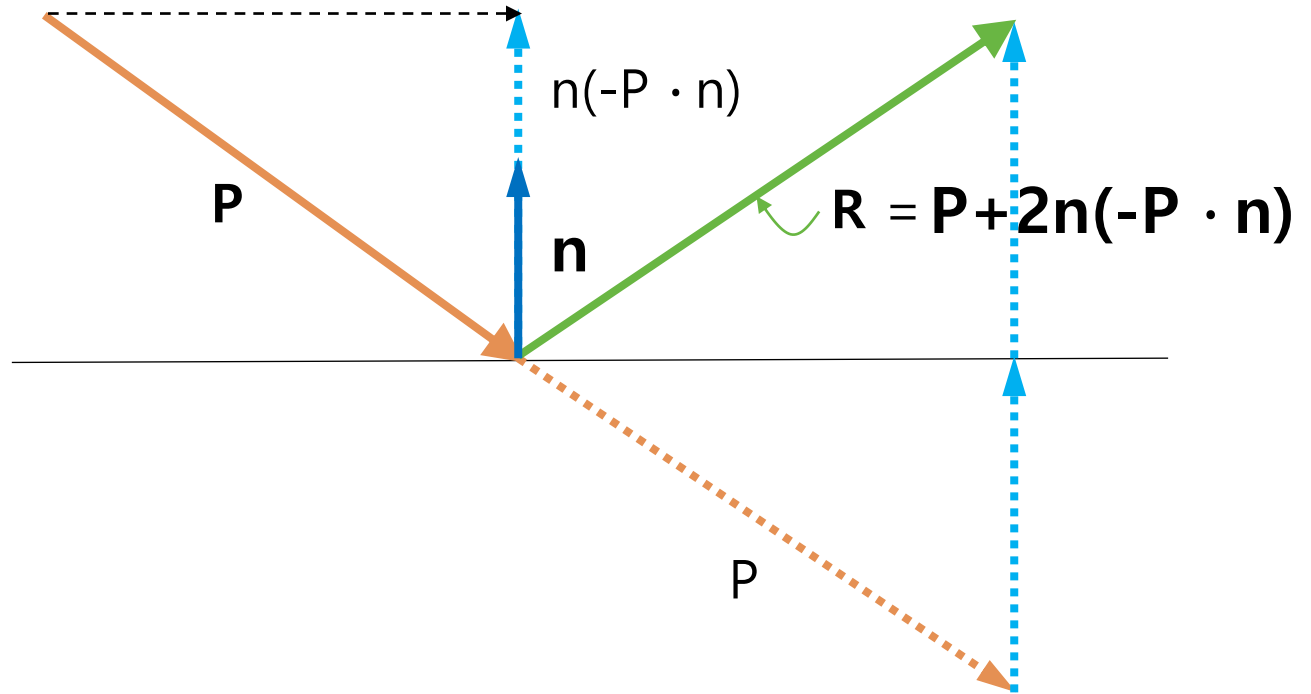
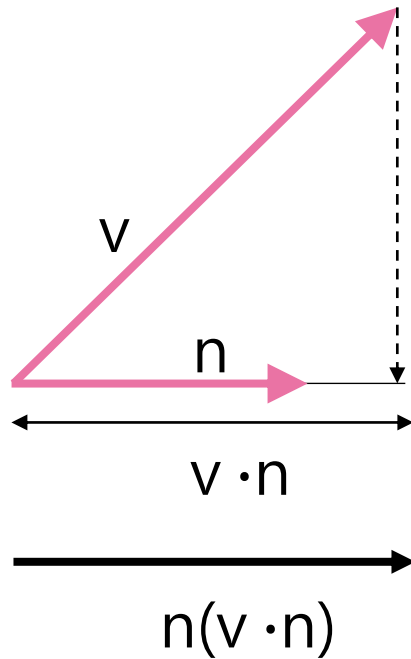


Vector3.Reflect

- 법선에 의해 정의된 평면에서 벡터 반사
- `Vector3.Reflect(Vector3 inDirection, Vector3 inNormal);`



입사벡터와 반사벡터



내적은 스칼라값이므로 방향벡터를 곱해 벡터로 변환

Physics.Raycast() 함수

```
public static bool Raycast(Vector3 origin, Vector3 direction, float maxDistance = Mathf.Infinity,
    int layerMask = DefaultRaycastLayers,
    QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);
```

```
public static bool Raycast(Vector3 origin, Vector3 direction, out RaycastHit hitInfo,
    float maxDistance, int layerMask, QueryTriggerInteraction queryTriggerInteraction);
```

파라미터	내용
origin	Ray의 시작점
direction	Ray의 방향
hitInfo	true가 반환되면 hitInfo는 충돌한 콜라이더에 대한 정보를 포함 (참조 : RaycastHit).
maxDistance	Ray가 충돌여부를 확인해야하는 지점까지의 최대거리
layerMask	레이 캐스팅 시에 Collider를 선택적으로 무시하기 위해 사용
queryTriggerInteraction	해당 query가 Trigger를 hit 해야 하는지에 대해 알림

Ray 구조체

Ray는 origin에서 시작해서 direction 방향으로 나아가는 무한대 길이의 선

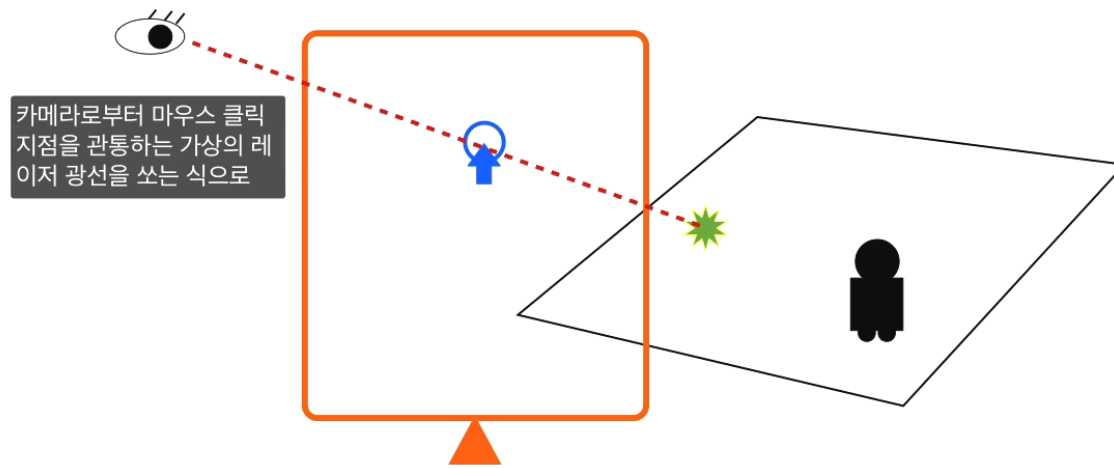
변수	내용
origin	Ray의 시작점
direction	Ray의 방향

함수	내용
GetPoint()	Ray의 Distance 거리의 지점을 반환

마우스 클릭 이동의 원리



2차원 좌표에서 3차원 위치 파악



LineRenderer

```
[RequireComponent(typeof(LineRenderer))]
```

```
LineRenderer lr;
```

```
void Start() {
```

```
    Ray ray = new Ray(transform.position, transform.forward);
```

```
    lr = GetComponent<LineRenderer>();
```

```
    lr.SetPosition(0, transform.position);
```

```
    lr.SetPosition(1, ray.GetPoint(30.0f));
```

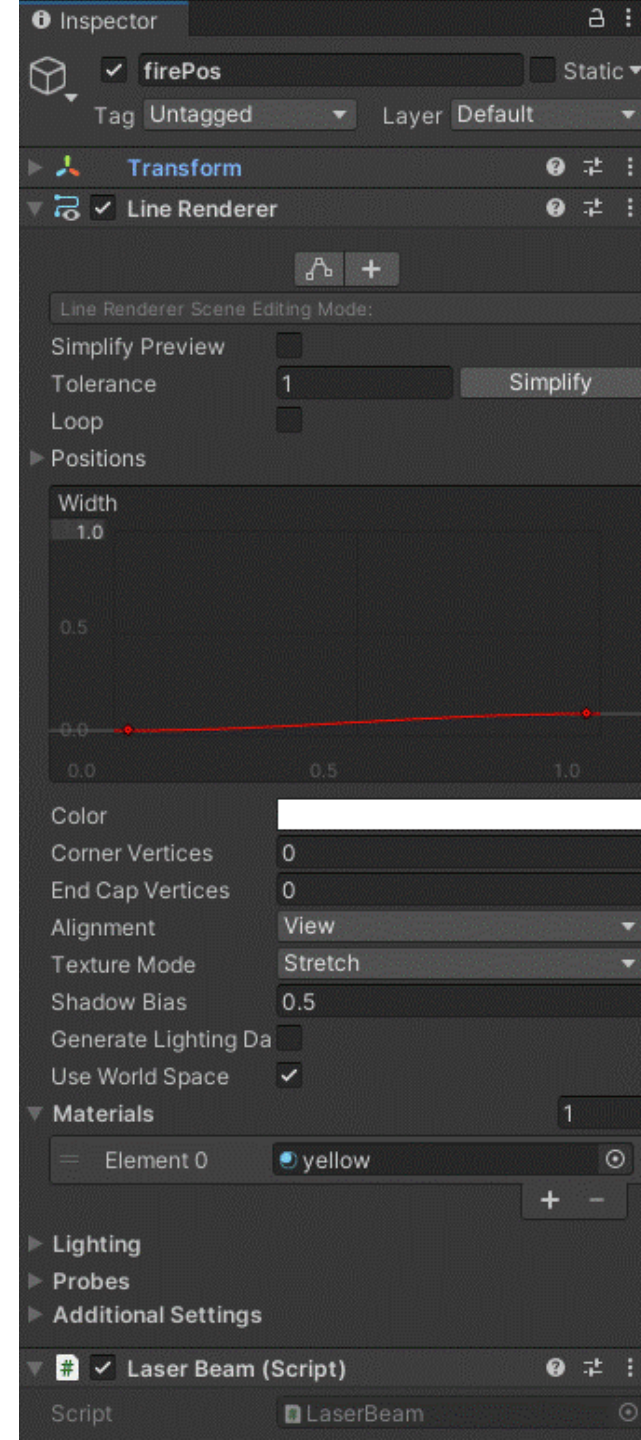
```
}
```



```

[RequireComponent(typeof(LineRenderer))]
public class LaserBeam : MonoBehaviour{
    LineRenderer lazer;
    void Start() {
        lazer = GetComponent<LineRenderer>();
        lazer.enabled = false;
    }
    void Update() {
        Ray ray = new Ray(transform.position, transform.forward);
        RaycastHit hit;
        if (Input.GetMouseButtonDown(0)) {
            lazer.SetPosition(0, ray.origin);
            if (Physics.Raycast(ray, out hit, 100.0f)) {
                lazer.SetPosition(1, hit.point);
            }
            else {
                lazer.SetPosition(1, ray.GetPoint(100.0f));
            }
            StartCoroutine>ShowLaserBeam();
        }
    }
    IEnumerator ShowLaserBeam() {
        lazer.enabled = true;
        yield return new WaitForSeconds(Random.Range(0.01f, 0.2f));
        lazer.enabled = false;
    }
}

```



```
private LineRenderer lr;  
private void Awake() {  
    lr = GetComponent<LineRenderer>();  
}  
private void Update() {  
    lr.positionCount = 1;  
    lr.SetPosition(0, transform.position);  
    Ray ray = new Ray(transform.position, transform.forward);  
    RaycastHit hit;  
    for (int i = 0; i < 10; i++) { // 최대 10회만 반사  
        if (Physics.Raycast(ray, out hit)) { // 거울과 충돌하면  
            lr.positionCount++;  
            lr.SetPosition(lr.positionCount - 1, hit.point);  
            ray = new Ray(hit.point, Vector3.Reflect(ray.direction, hit.normal)); // 반사된 광선 생성  
        }  
        else {  
            lr.positionCount++;  
            lr.SetPosition(lr.positionCount - 1, ray.origin + (ray.direction * 100));  
            break;  
        }  
    }  
}
```

The background is a complex, abstract geometric pattern composed of numerous triangles in various colors including yellow, orange, red, pink, purple, blue, and green. These triangles are arranged in a way that creates a sense of depth and movement. A white rectangular box is positioned in the lower-middle section of the image, containing the text 'C# 과제 풀이'. Below this box is a solid pink horizontal bar.

C# 과제 풀이

- 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)

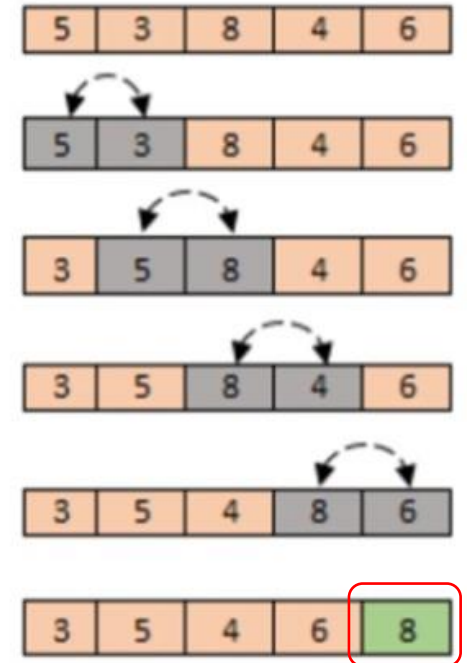
회문의 예(한글)	회문의 예(영어)
역삼역	mom
기러기	wow
일요일	noon
사진사	level
복불복	radar
다가가다	kayak
기특한 특기	racecar
다했나? 했다!	God's dog
다시 합창 합시다	Madam, I'm Adam.



버블 정렬 :: $O(n^2)$

```
static void Main(string[] args)
{
    int[] nums = { 33, 115, 9, 53, 4, 6 };
    sort(ref nums);
}

public static void sort(ref int[] dataArray)
{
    for (int i = 0; i < dataArray.Length - 1; i++)
    {
        for (int j = 0; j < dataArray.Length - 1 - i; j++)
        {
            if (dataArray[j] > dataArray[j + 1])
            {
                int temp = dataArray[j];
                dataArray[j] = dataArray[j + 1];
                dataArray[j + 1] = temp;
            }
        }
    }
}
```



완료

삽입 정렬 : $O(n^2)$, Best: $O(n)$

```
static void Main(string[] args)
{
    List<int> numList = new List<int>() { 2, 42, 5, 17, 3 };

    List<int> temp = InsertionSort(numList);

    foreach (var item in temp)
    {
        Console.WriteLine(item);
    }
}
```

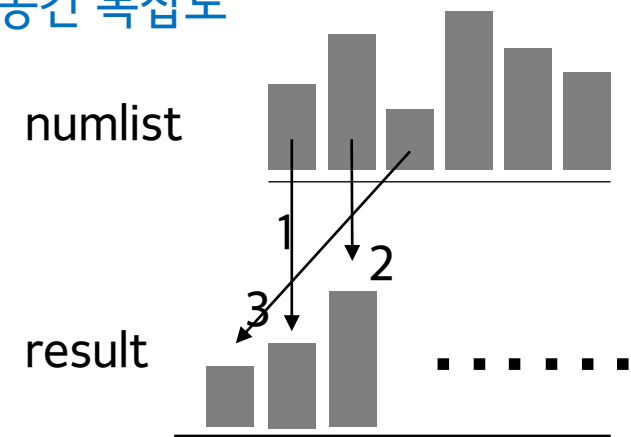
```
// 들어갈 위치 i 반환
private static int findInsIdx(List<int> result, int value)
{
    for (int i = 0; i < result.Count; i++)
    {
        if(value < result[i])
        {
            return i;
        }
    }
    return result.Count;
}
```

```
private static List<int> InsertionSort(List<int> numList)
{
    List<int> result = new List<int>();

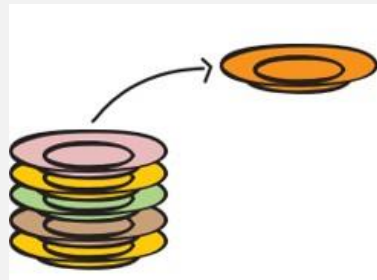
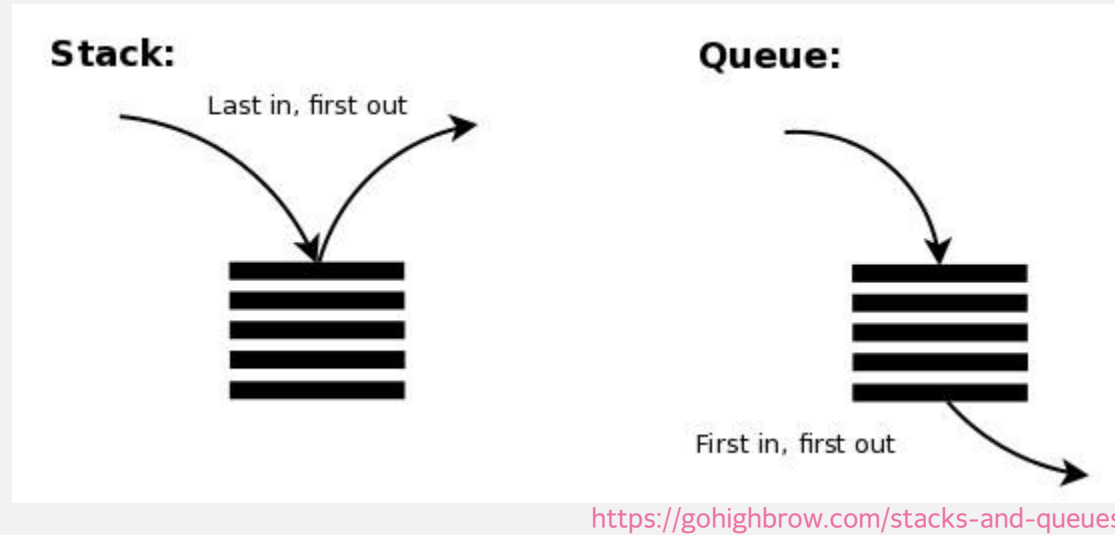
    while (numList.Count != 0)
    {
        int value = numList[0];
        numList.RemoveAt(0); //맨 앞을 뽑아냄

        int idx = findInsIdx(result, value);
        result.Insert(idx, value);
    }
    return result;
}
```

시간 복잡도와 공간 복잡도

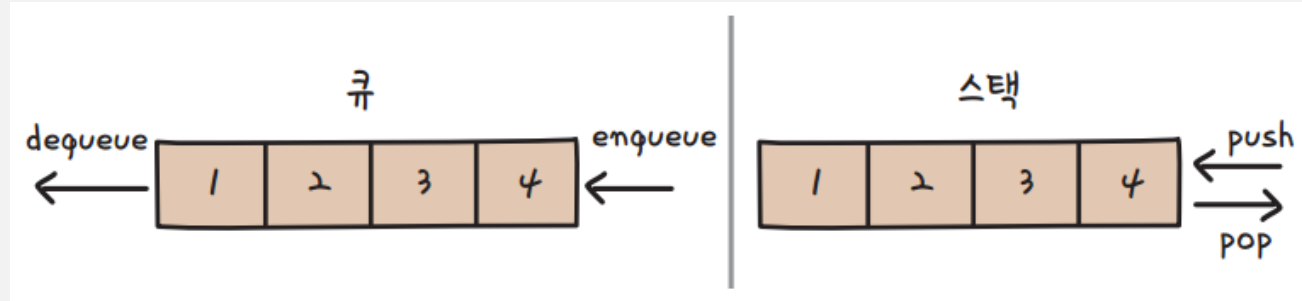


- 들어간 자료가 일렬로 보관되며 자료를 넣는 동작과 빼는 동작 가능
- **큐** : 택시줄서기, **스택** : 접시 쌓기



큐와 스택

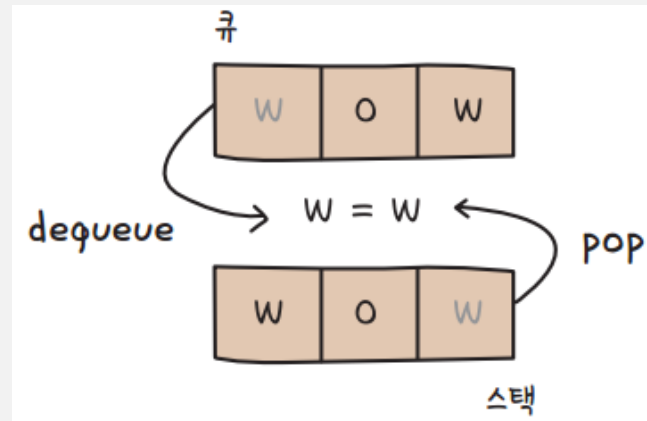
```
Queue queue = new Queue();  
Stack stack = new Stack();
```



- Enqueue() : 큐에 자료를 한 개 집어넣는 동작
- Dequeue() : 큐 안에 있는 자료를 한 개 꺼내는 동작
- Push() : 스택에 자료를 하나 집어넣는 동작
- Pop() : 스택 안에 있는 자료를 하나 꺼내는 동작

회문 찾기 알고리즘

- 주어진 문자들을 하나씩 큐와 스택에 넣음
- 큐에서 꺼낸 문자들(원래 순서)이 스택에서 꺼낸 문자들(역순)과 모두 같다면 그 문장은 회문



Palindrome

```
static void Main(string[] args)
{
    Console.WriteLine(Palindrome("WOW"));
    Console.WriteLine(Palindrome("우영우"));
    Console.WriteLine(Palindrome("Madam, I'm Adam"));
}

public static bool IsAlpha(string input)
{
    return Regex.IsMatch(input, "[a-zA-Z가-힣]");
}
```

정규식 (regular expression)

```
private static object Palindrome(string sentence)
{
    Stack stack = new Stack();
    Queue queue = new Queue();

    for (int i = 0; i < sentence.Length; i++)
    {
        if (IsAlpha(sentence[i].ToString()))
        {
            stack.Push(sentence[i]);
            queue.Enqueue(sentence[i]);
        }
    }

    while (queue.Count != 0)
    {
        string a = queue.Dequeue().ToString().ToLower();
        string b = stack.Pop().ToString().ToLower();

        Console.WriteLine($"{a}, {b}");

        if (a != b)
        {
            return false;
        }
    }

    return true;
}
```

Palindrome2

```
private static bool Palindrome2(string sentence)
{
    string s = "";
    for (int i = sentence.Length-1; i >= 0; i--)
    {
        s += sentence[i].ToString();
    }
    Console.WriteLine($"{sentence}, {s}");

    if(sentence == s)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

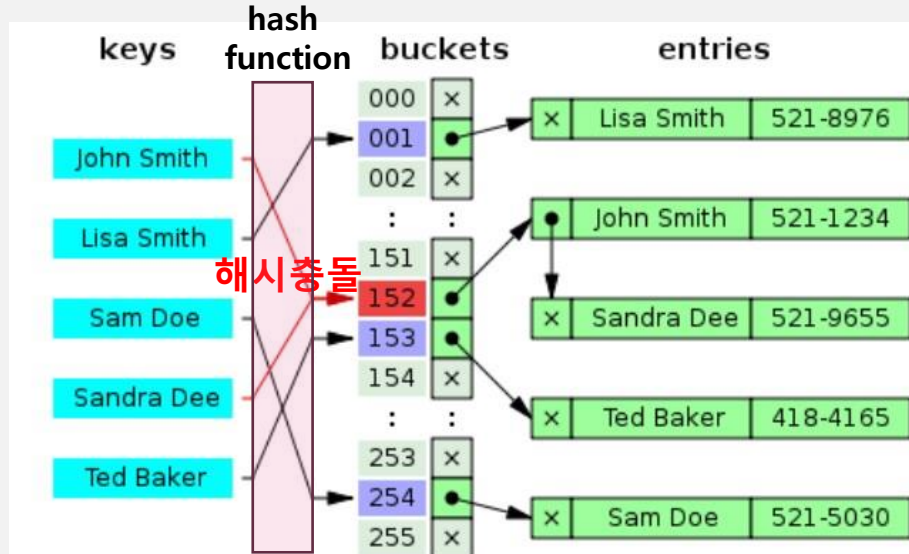
문장의 뒤에 있는 글자 부터
꺼내서 새로운 문자열을 만듦

Vector2Int (Unity)

- 2차원 정수 벡터를 표현하는 구조체
- 그리드에서 좌표의 표현이 쉽고 연산이 단순화 됨
- Vector2와 비교하여 메모리 사용량이 적음 (많은 정수 좌표 관리에 좋음)
- `Vector2Int point = new Vector2Int(0, 0)`




- **해시함수** : 임의의 길이의 데이터를 고정된 크기의 값(해시 값)으로 변환하는 함수
 - 주어진 결과값에 대해 입력값을 계산하기 어려워야 한다.
 - 하나의 주어진 입력에 대하여 같은 출력으로 사상시키는 또 다른 입력을 찾는 것이 어려워야 한다.
- **해시테이블**: 해시함수를 사용하여 키를 해시값으로 매핑하고, 이 해시값을 색인(index) 혹은 주소 삼아 데이터의 값(value)을 키와 함께 저장하는 **자료구조**




해시값을 색인(index)에 사용함으로써
검색과 삽입/삭제를 빠르게 수행 : $O(1)$

hashSet (C#)

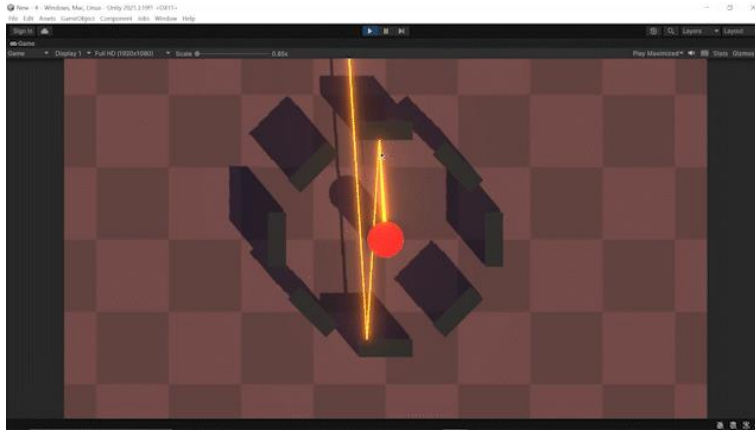
- 집합처럼 중복을 허용하지 않는 데이터 구조
- 특정 인덱스에 추가/제거 불가, 순서를 보장 불가, 정렬 불가
- 추가, 검색, 삭제시 시간복잡도가 모두 $O(1)$ Vs. List = 추가: $O(1)$ 검색, 삭제시 $O(n)$
- 해시테이블을 사용해 해당요소를 빠르게 가져올 수 있음
- List와 유사하게 Add, Contains, Clear 메서드와 Count 속성이 있음
- 집합을 다루는 유용한 메서드 제공
 - [UnionWith](#), [IntersectWith](#), [IsSubsetOf](#), [IsSupersetOf](#), etc
- Dictionary<TKey, TValue>와 비슷하지만 키 부분만 저장하고 값은 저장하지 않음
- **HashSet<int> numbers = new HashSet<int>();**



C# 과제



- (필수) Mirror reflection 안보고 만들어보기
- Vector3.Reflect()를 사용하지 않고 구현하기



- (선택) BFS를 이용해 미로찾기 알고리즘 문제 해결