

# 자료구조와 알고리즘

2강 - 배열과 리스트

LECTURED BY SOONGU HONG



# 1. 배열과 리스트 특징

## \* 배열 (ARRAY)

인덱스	0	1	2	3	4	5
	값1	값2	값3	값4	값5	값6

- 배열은 메모리의 연속 공간에 값이 채워져 있는 형태의 자료구조입니다.
- 배열의 값은 인덱스를 통해 참조할 수 있으며, 선언한 자료형의 값만 저장할 수 있습니다.

## \* 배열의 특징

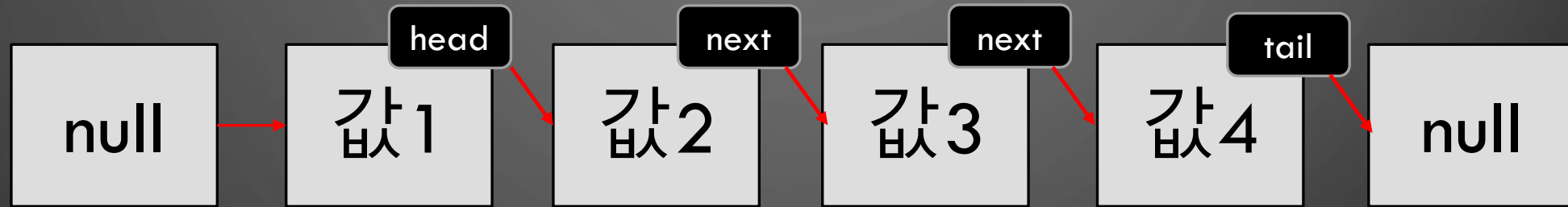
인덱스를 사용하여 특정 값에 바로 접근할 수 있다.

새로운 값을 삽입하거나 특정 인덱스에 있는 값을 삭제하기 어렵다.  
값을 삽입하거나 삭제하려면 해당 인덱스 주변에 있는 값을 이동시키는 과정이 필요하다.

배열의 크기는 선언할 때 지정할 수 있으며, 한 번 선언하면 크기를 늘리거나 줄일 수 없다.

구조가 간단하므로 코딩 테스트에서 많이 사용한다.

## \* 리스트 (LIST)

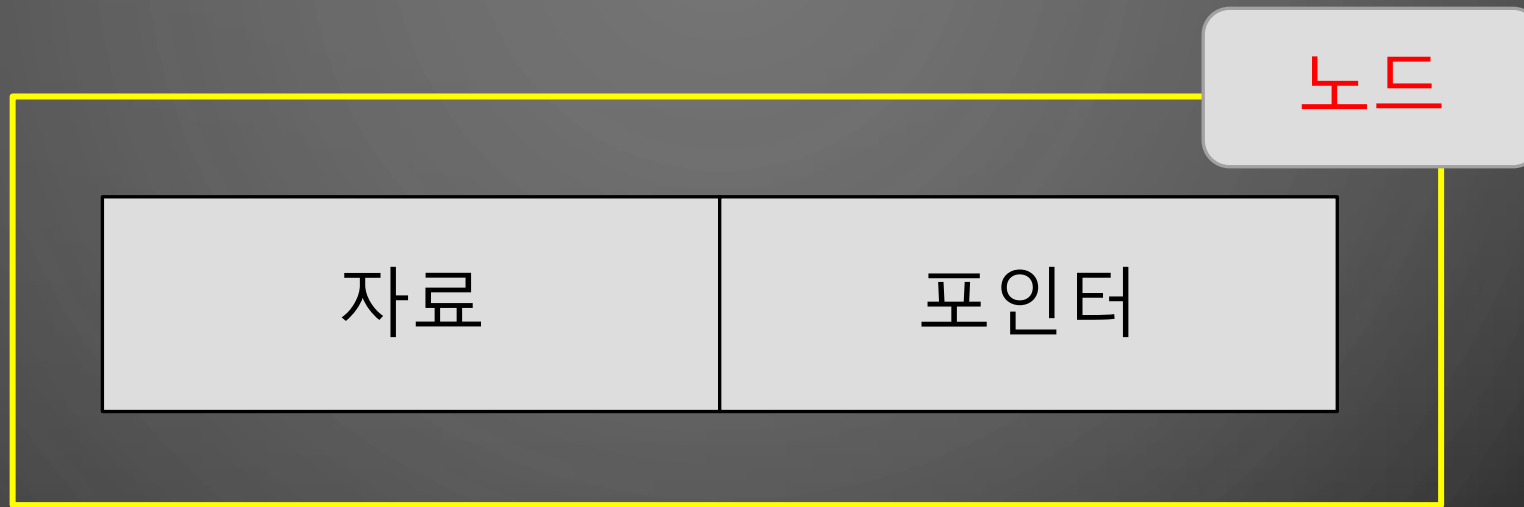


- 리스트는 값과 포인터를 묶은 노드라는 것을 포인터로 연결한 자료구조입니다.

## \* 연결 리스트 (LINKED-LIST)

- 연결 리스트는 프로그래밍에서 사용되는 가장 기초적인 자료구조 중 하나입니다.
- 일정한 순서를 가지는 자료 요소들을 표현하는 방법으로 일련의 자료 요소들을 통합하여 관리함으로써 정보의 추적과 탐색을 효율적으로 실현합니다.
- 사실상 리스트 구현을 위해서 배열보다는 연결 리스트를 많이 사용하므로 리스트와 연결리스트를 동의어로 많이 사용합니다.

## \* 연결 리스트 - 노드(NODE)



- 저장된 자료와 함께 다음 자료를 연결하기 위한 포인터가 저장되어 있는 객체를 노드라고 부릅니다.

## \* 연결 리스트의 장단점

장점	단점
<ul style="list-style-type: none"><li>- 자료의 삽입과 삭제가 용이하다.</li><li>- 리스트 내에서 자료의 이동이 필요하지 않다.</li><li>- 사용 후 기억 장소의 재사용이 가능하다.</li><li>- 연속적인 기억 장소의 할당이 필요하지 않다.</li></ul>	<ul style="list-style-type: none"><li>- 포인터의 사용으로 인해 저장 공간의 낭비가 있다.</li><li>- 알고리즘이 복잡하다</li><li>- 특정 자료의 탐색 시간이 많이 소요된다.</li></ul>



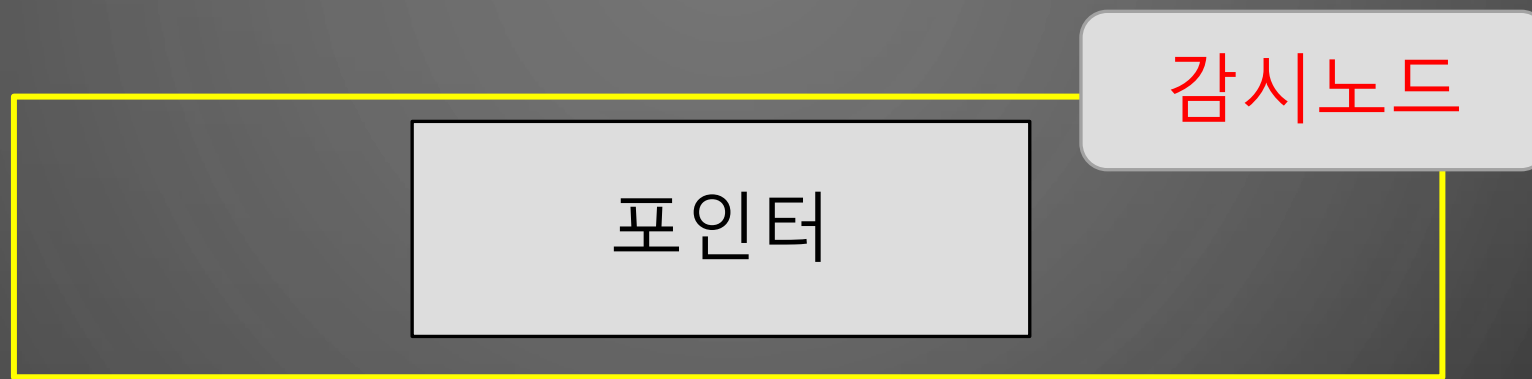
## \* 배열과 연결 리스트의 복잡도 비교

	배열	연결 리스트
인덱싱*	$O(1)$	$O(N)$
자료 삽입	$O(N)$	$O(1)$
자료 삭제	$O(N)$	$O(1)$
지역성**	좋다	나쁘다


\* 인덱싱이란 리스트 내에 존재하는 자료들 중 임의로 하나를 선택하는 연산을 말함.

\*\* 지역성은 자료들이 저장되는 메모리에 얼마만큼 밀집되어 있는지를 말함.  
밀집되어 있을수록 지역성이 좋으며 탐색속도가 빠름.

## \* 연결 리스트 – 감시노드(SENTINEL-NODE)



- 첫번째 자료를 가진 노드는 두번째 자료의 위치를 포인터로 기억하고 있습니다. 또한 두번째 자료를 가진 노드는 세번째 자료의 위치를 알고 있습니다.
- 그러면 첫번째 자료의 위치는 누가 알고 있을까요?
- 이 문제를 해결하기 위한 개념이 감시노드입니다.
- 감시노드는 헤더노드라고 부르기도 하며, 첫번째 노드의 위치만을 기억하여 연결리스트의 시작점을 기억합니다.



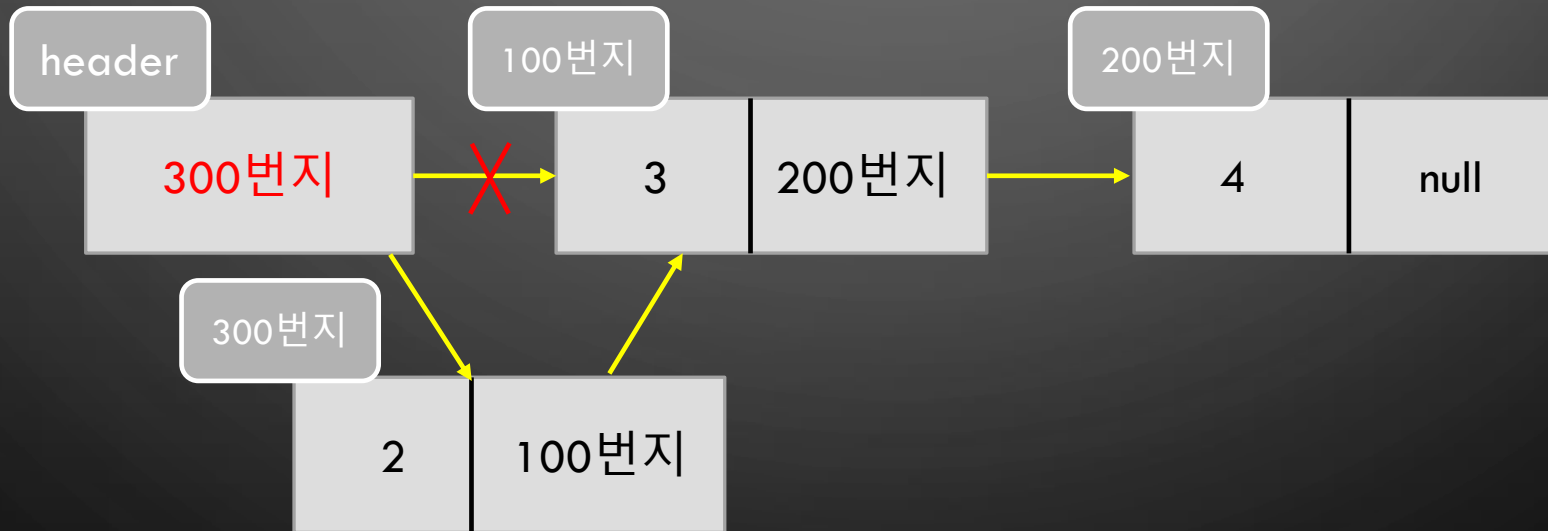
## 2. 단방향 연결리스트 (SINGLY LINKED-LIST)

## \* 단순 연결 리스트(SIMPLE LINKED-LIST)



- 단순 연결리스트는 연결 리스트의 가장 단순한 형태입니다.
- 각 노드들은 오직 하나의 포인터만 갖고 있기 때문에 바로 자신의 다음 노드만 참조할 수 있습니다.
- 마지막 노드의 포인터가 null값을 가리킬 경우 리스트의 끝을 의미하며 header가 null값일 경우 빈 리스트를 나타냅니다.

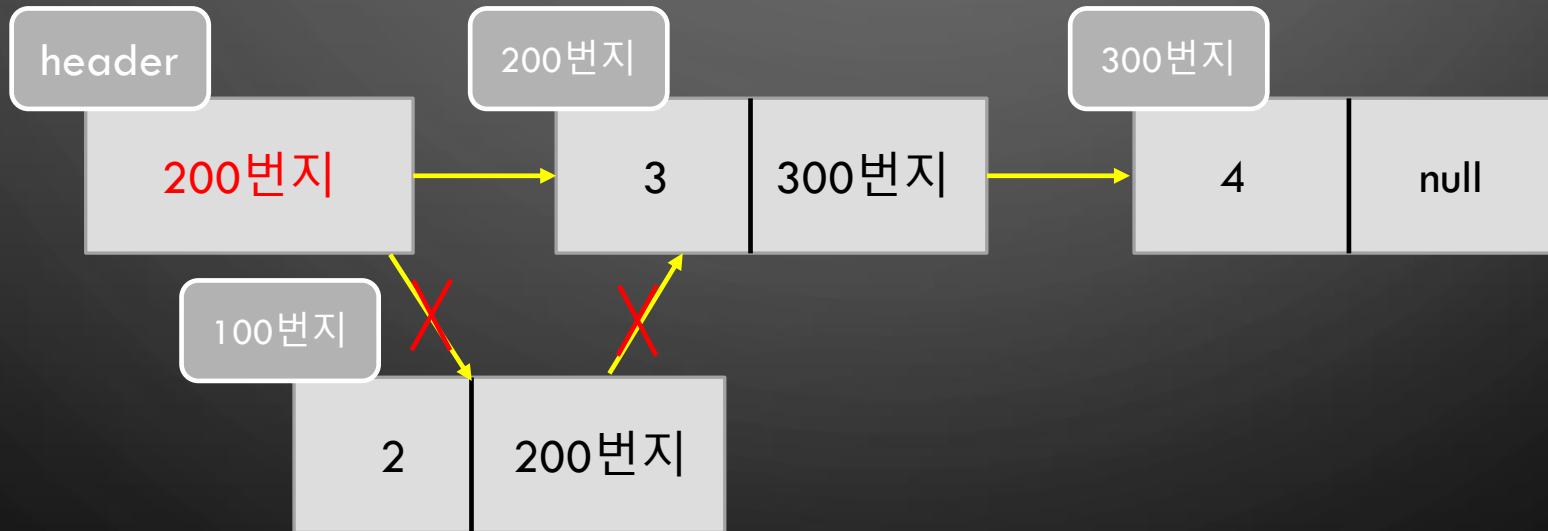
## \* 첫번째 위치에 자료 삽입하기



## \* 첫번째 위치에 자료 삽입하기

- 자료를 맨 첫 위치에 삽입하는 경우에는 헤더노드의 포인터만 새롭게 삽입할 노드의 주소로 변경해주면 되기 때문에 간단합니다.
- 즉, 헤더는 새로 삽입된 노드의 주소를 기억하고, 새로 삽입된 노드는 기존의 첫번째 노드의 주소를 포인터에 기억하면 됩니다.
- 단, 이런 기능은 정렬된 리스트에서는 사용하면 안됩니다.  
정렬된 리스트란 삽입과 동시에 자료들이 정렬되는 리스트를 말하는데 이런 상태를 지속적으로 유지하려면 강제로 첫번째 위치에 삽입하는 기능을 만드는 것은 위험합니다.
- 이 기능은 스택 구현 시 유용하게 사용할 수 있으니 잘 기억해두세요.

## \* 첫번째 위치 자료 삭제하기



## \* 첫번째 위치에 자료 삭제하기

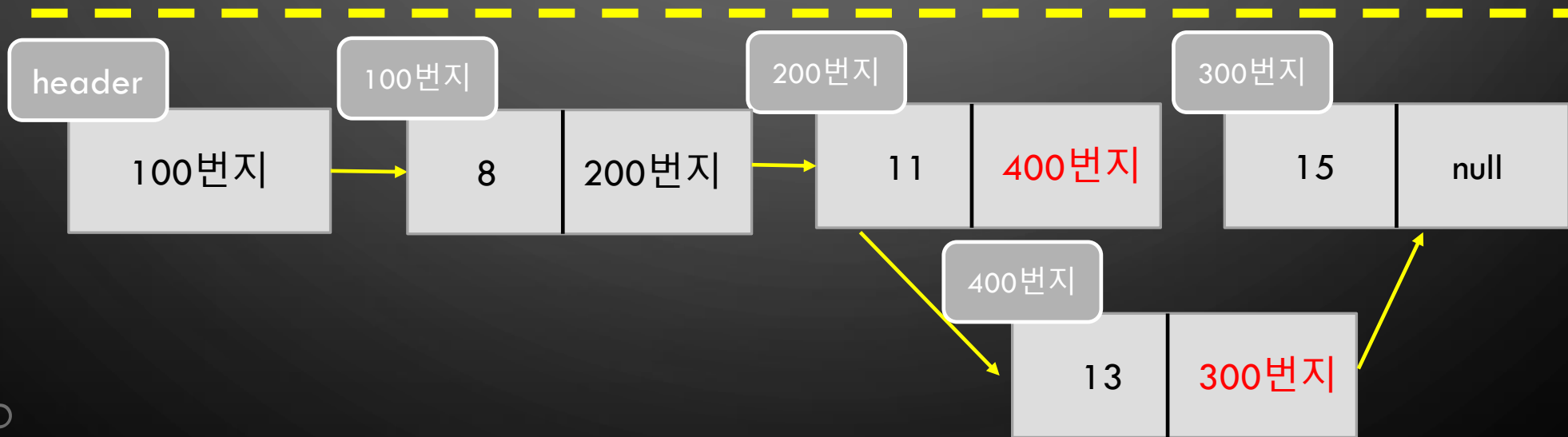
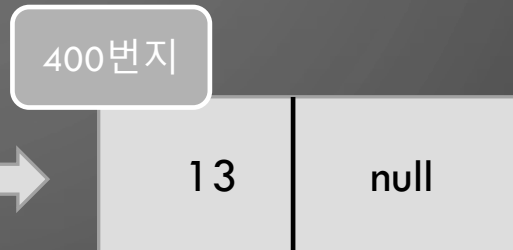
- 첫번째 위치 자료 삭제기능은 첫번째 삽입과 상반되는 구조입니다.
- 헤더의 포인터를 첫번째가 아닌 기존의 두번째 노드의 주소를 기억하게 만들면 됩니다.
- 단, 빈 리스트 상태라면 삭제할 노드가 존재하지 않으므로 if문을 통해 검사합니다.
- 그러면 삭제되는 노드에 대한 처리는 메모리 안에 존재할 텐데 어떻게 할까요?
- c언어나 c++에서는 프로그래머가 해당 처리도 직접 해줘야 메모리 누수(leak)을 막을 수 있습니다. 하지만 자바에서는 쓰레기 수집기(garbage collection) 프로세스가 알아서 연결이 끊어진 노드를 정리해주므로 좀 더 구현이 수월합니다.



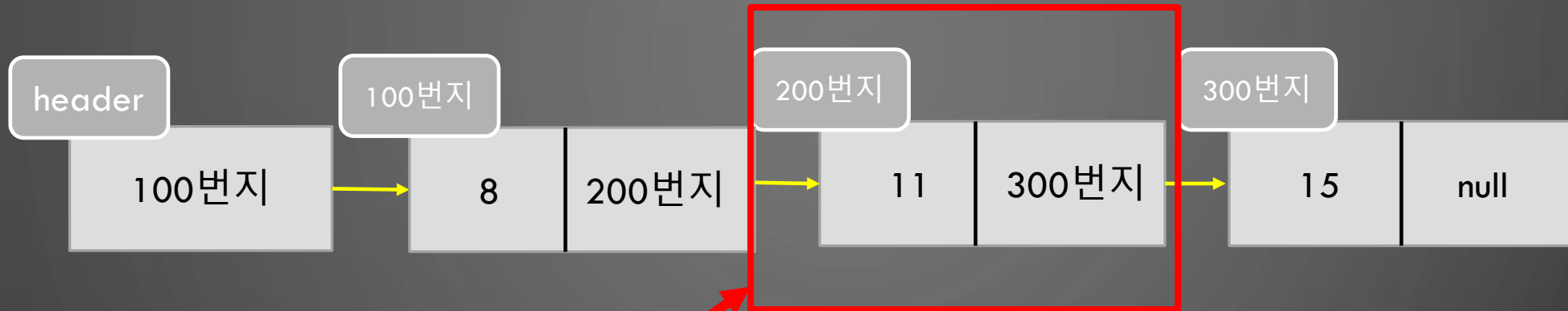
# \* 정렬된 리스트에서 특정 위치 자료 삽입하기



자신이 들어갈  
위치를 탐색




# \* 정렬된 리스트에서 특정 위치 자료 삭제하기



삭제대상





### 3. 양방향 연결리스트 (DOUBLY LINKED-LIST)

## \* 이중 연결 리스트(DOUBLE LINKED-LIST)



- 단순 연결리스트의 단점 중 하나는 단방향성입니다. 즉, 한방향으로만 노드들을 탐색할 수 있다는 의미입니다.
- 이중 연결 리스트는 양방향성을 가지고 있어서 순방향, 역방향 탐색이 모두 가능합니다.