List

- 값(value)들을 저장하는 추상자료형(ADT)
- 순서가 있음
- 중복을 허용

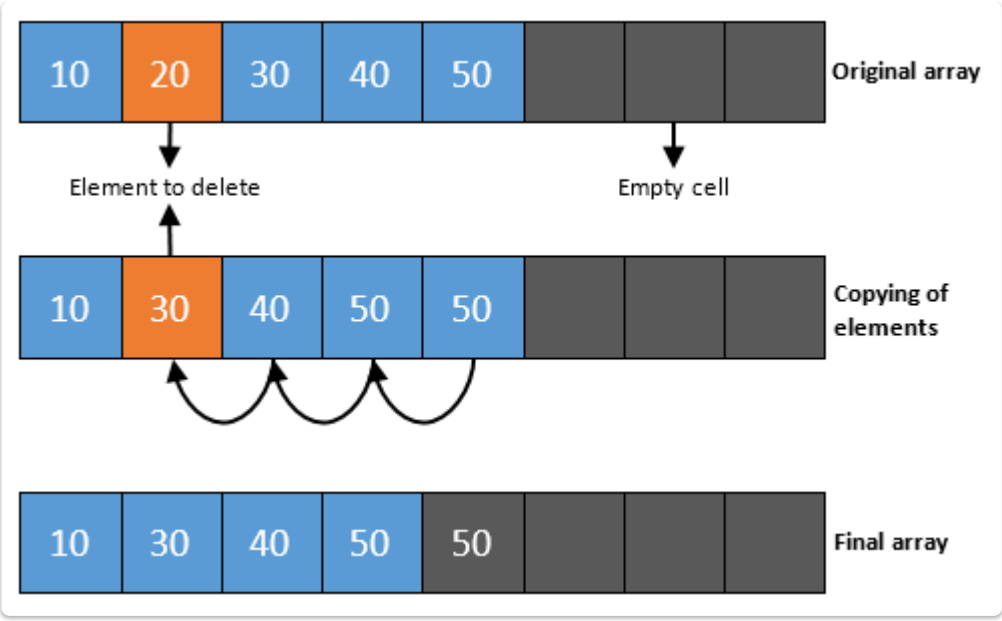
💡 List는 언제 쓸까 ?

- Set이나 Map을 사용하는게 더 적절한 상황이 아니라면, 거의 대부분 List를 사용한다고 봐도 무리 없습니다.

📌 List 구현체

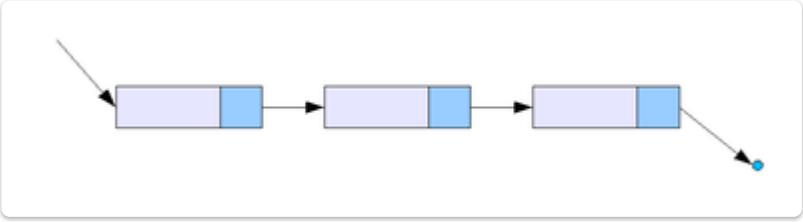
💡 Array List

- 배열(array)을 사용하여 List를 구현
- 장점
  - 데이터의 참조가 쉽다. 인덱스 값을 기준으로 어디든 한 번에 참조가 가능하다.
  - 배열을 선언할 때 정해진 크기만큼 연속된 메모리 주소를 할당 받는다. 이 덕분에 임의의 인덱스로 임의 접근(random access)이 가능하다.
- 단점
  - 연속된 메모리 주소를 할당해야 하기 때문에 배열의 길이가 초기에 결정되어야 한다. 즉, 변경이 불가능하다.
  - 삭제의 과정에서 데이터의 이동(복사)가 매우 빈번히 일어난다.



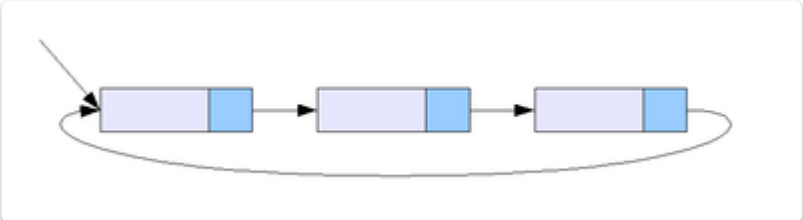
위의 그림처럼 배열에서 특정 값을 삭제하게 된다면 그 값 이후의 값들을 전부 한 칸씩 앞으로 옮겨야 한다. 그렇기 때문에 많은 시간이 소요된다.

💡 Linked List



- 노드를 연결(linked)시키는 형태로 구현
- 장점
  - 데이터의 추가와 삭제가 용이하다.
  - 리스트의 크기를 자유롭게 조절할 수 있다. 즉, 크기를 미리 지정할 필요가 없다.
- 단점
  - 배열과 달리 연속적인 메모리 주소를 할당 받지 않기 때문에 임의 접근이 불가능하다.
  - 대신 연결 리스트에서는 탐색을 위해 순차 접근(sequential access) 방식을 사용한다.

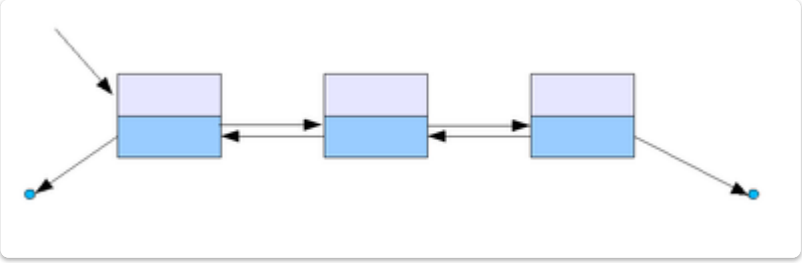
1. Circular linked list



- tail 이 가리키고 있는 가장 마지막 노드가 가장 첫 노드를 가리키고 있는 경우

- tail 과 head가 연결되어 있는 경우

2. Doubly linked list



- 양방향 통행
- `prev`, `next` 를 모두 저장하여 앞, 뒤로 모두 이동 가능.

3. Circular Doubly linked list

- 위 두 개를 합친 경우

💡 ArrayList vs LinkedList

	Array list	Linked list
구현	배열(array) 사용	노드를 연결(linked)
데이터 접근 시간	모든 데이터 상수 시간 접근	위치에 따라 이동 시간 발생
삽입/삭제 시간	삽입/삭제 자체는 상수 시간	
	삽입/삭제 시 데이터 시프트가 필요한 경우 추가 시간 발생	삽입/삭제 위치에 따라 그 위치까지 이동하는 시간 발생
리스트 크기 확장	배열 확장이 필요한 경우 새로운 배열에 복사하는 추가 시간 발생	-
검색	최악의 경우 리스트에 있는 아이템 수 만큼 확인	
CPU cache	CPU cache 이점 활용	-
구현 예	Java의 ArrayList, CPython의 list	Java의 LinkedList

1. 데이터 탐색

- ArrayList
  - 임의의 접근 방식을 사용하기 때문에 인덱스를 이용해 바로 탐색할 수 있다.

    - `O(1)`
- LinkedList
    - 순차 접근 방식을 사용하기 때문에 어떤 데이터를 찾기 위해서는 처음부터 순차적으로 탐색해야 한다.
    - `O(n)`

2. 데이터 추가

- ArrayList
    - 데이터들이 순차적으로 저장되어 있기 때문에 맨 처음이나 그 이후에 데이터를 추가하려면 그 뒤에 있는 데이터들을 전부 한 칸씩 뒤로 옮겨야 한다.
    - `O(n)`
    - 맨 뒤에 추가할 때는 다른 데이터들의 위치를 옮길 필요가 없으므로 `O(1)` 이다.
- LinkedList
    - 데이터를 추가하는 행위 자체의 시간 복잡도는 `O(1)`.
    - 데이터를 추가하는 위치가 맨 앞이 아니라면 그 위치까지 순차적으로 탐색하면서 이동해야 하므로 추가 위치까지 이동할 때의 시간 복잡도는 `O(n)`.
    - 따라서 추가하려는 데이터의 위치가 맨 앞이라면 `O(1)`, 아니라면 `O(n)`.

3. 데이터 삭제

- ArrayList
    - 삭제하려는 데이터의 위치가 맨 뒤가 아니라면 `O(n)`.
    - 삭제하려는 데이터의 위치가 맨 뒤라면 `O(1)`.
- LinkedList
    - 삭제하려는 데이터의 위치가 맨 앞이라면 `O(1)`.
    - 삭제하려는 데이터의 위치가 맨 앞이 아니라면 `O(n)`.

💡 결론

- ArrayList 는 LinkedList 와 비교했을 때 상대적으로 데이터의 접근과 탐색에 우위를 가진다.
- LinkedList 는 ArrayList와 비교했을 때 상대적으로 데이터의 추가와 삭제에 우위를 가진다.
- 데이터의 접근과 탐색이 중요한 경우엔 ArrayList, 데이터의 추가와 삭제가 중요한 경우엔 LinkedList 를 사용하면 된다.