리스트

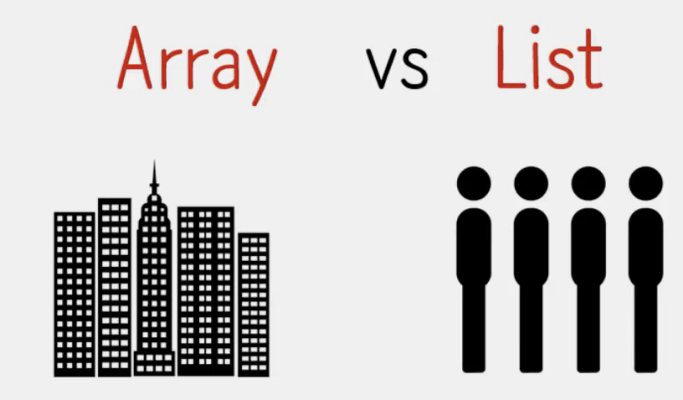
작성일시	@June 5, 2023 6:48 PM
자료	https://opentutorials.org/module/1335/8636 https://opentutorials.org/module/1335/8821
#주차	2

> **ADT(Abstract Data type)**?
> 예로 들어, 일상생활에서 장보기 리스트를 작성하여 사용한다고 할 때, 이를 활용하여 컴퓨터에서도 사용하려면 이것을 구현하는데 필요한 수행 작업등을 정의하는 것(일종의 설계도) 하지만 이것은 실제 구현을 위한 데이터가 어떻게 구성되고 어떤 알고리즘이 사용되는지 명시하는 것 아님

데이터 스트럭처에서 가장 중요한 것은 그 데이터 스트럭처가 어떤 기능을 가지고 있고 어떻게 동작할 것인지다.

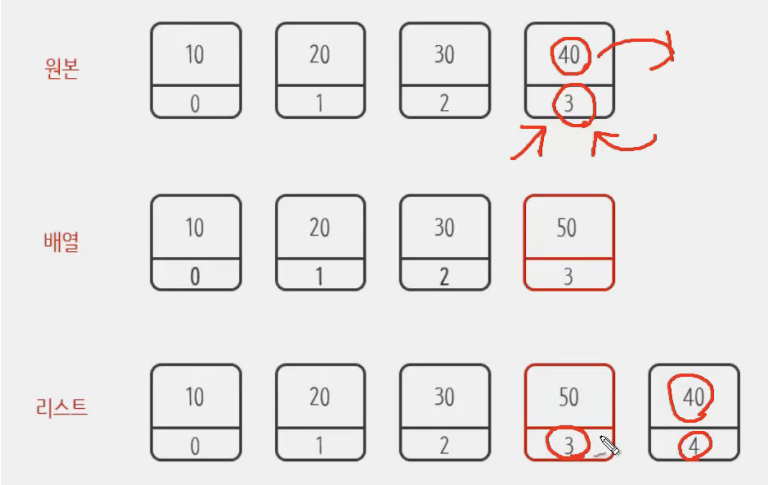
데이터 스트럭처의 타입을 결정하는 것은 그 데이터의 오퍼레이션(즉, 기능)이다

배열 vs 리스트



- **배열**의 가장 큰 특징은 **인덱스**가 있다는 것. 인덱스를 이용해 특정 데이터를 접근하고 가져올 수 있음
  배열의 인덱스는 그 값에 대한 유일무이한 식별자라고 할 수 있다.
- **리스트**도 내부적으로 인덱스를 가지고 있을 순있지만 가장 중요한 것은 현재 데이터의 다음은 이것이고 이것 다음 데이터는 이것이라는 데이터가 저장된 **순서**가 훨씬 중요
  리스트의 인덱스는 몇번째 데이터인가 정도의 의미만 가지고 있음

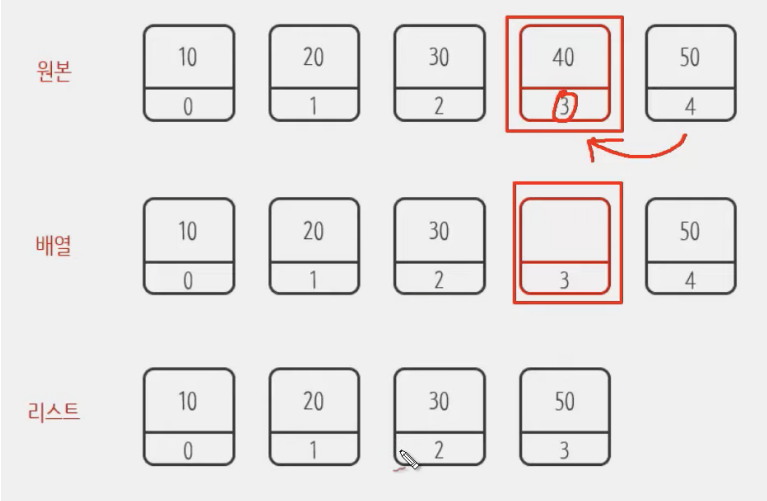
추가



만약 인덱스 3번 위치에 데이터를 추가한다고 하면

- 배열 → 기존의 40은 사라지고 추가하려고 하는 데이터가 덮어씌기 됨
- 리스트→ 기존의 40을 뒤로 한 뒤로 미루고 그 자리에 새로 데이터를 추가함

삭제



만약 인데스 3번 위치에 데이터를 삭제한다고 하면

- 배열 → 해당위치의 값이 삭제되고 인덱스 3번은 비어져 있음
- 리스트 → 인덱스 3번위치 데이터가 삭제되면서 해당 데이터 뒤에 있던 데이터가 한 칸 앞으로 전진

이런 배열의 동작은 메모리의 낭비를 초래합니다. 또한 배열을 순회할 때 해당 위치에 데이터가 있는지 없는지 체크하는 로직이 필요하다는 의미이기도 함

리스트

- 목록, 순서가 있는 일련의 데이터 집합
- 리스트에 저장된 각 데이터 항목을 요소라함
- 프로그램 가용메모리가 리스트에 저장할 수 있는 최대 요소수
- 배열의 index를 버림(빠른 검색 X), 요소간의 순서가 중요
- 빈 엘리먼트를 허용하지 않음
- 데이터가 저장되어있는 위치(pos) 탐색을 통해 요소 추출
- **빈틈없는 데이터 적재**의 장점을 취한 자료구조
- 배열은 index자리에 값이 삭제되면 데이터가 떠버림 → 메우기 위한 로직이 필요
- 배열은 새로운 배열을 만들때 고정크기를 만들어야함 하지만 리스트는 고정된 크기가 아님(JS배열은 고정된 크기로 안만들어되기 때문에 헷갈릴 수 있음...)
- 리스트는 자료 크기 안정해져 있거나 데이터의 추가나 삭제가 빈번할 경우 사용하면 좋음
- 학생부를 관리하는 것을 배열(고정된 크기라 할 때)로 만들었다면, 새로운 전학생이 오면 새로 만들어야한다는 단점

리스트 ADT

리스트는 순서가 있는 일련의 데이터 집합이다. 리스트에 저장된 각 데이터 항목을 요소(element)라 부른다. 자바스크립트에서는 모든 데이터형이 리스트의 요소가 될 수 있다.

* 리스트의 가장 중요한 기능

프로퍼티	설명
pos	현재 위치
listSize	리스트의 요소수
length	리스트의 요소 수 반환
clear()	리스트의 모든 요소 삭제
toString()	리스트를 문자열로 표현해 반환
getElement()	현재 위치의 요소를 반환
insert() *	기존 요소 뒤로 새 요소를 추가
append() *	새 요소를 리스트의 끝에 추가
remove() *	리스트의 요소 삭제
front()	현재 위치를 리스트 첫 번째 요소로 설정
end()	현재 위치를 리스트 마지막 요소로 설정
prev()	현재 위치를 한 요소 앞으로 이동
next()	현재 위치를 한 요소 뒤로 이동
currPos()	리스트의 현재 위치를 반환
moveTo()	현재위치를 지정한 위치로 이동

List 클래스 구현

앞에서 정의한 리스트 ADT를 이용해 List 클래스를 구현할 수 있다.

```
function List() {
  this.dataStore = [];
  this.listSize = 0;
  this.pos = 0;
  this.length = function() {
    return this.listSize;
  };
  // 리스트의 다음 가용위치에 새 요소 추가 -> listSize 증가
  this.append = function(el) {
    this.dataStore[this.listSize++] = el;
  };
  this.remove = function(el) {
```

```
    const foundAt = this.find(el);
    if(foundAt > -1) {
      this.dataStore.splice(foundAt, 1);
      --this.listSize;
      return true;
    }
    return false;
  };
  this.find = function(el) {
    for(let i=0; i<this.dataStore.length; i++) {
      if(this.dataStore[i] === el) {
        return i;
      }
    }
    return -1;
  };
  this.toString = function() {
    return this.dataStore;
  };
  this.insert = function(el, after) {
    const insertPos = this.find(after);
    if(insertPos > -1) {
      this.dataStore.splice(insertPos+1, 0, el);
      ++this.listSize;
      return true;
    }
    return false;
  }
  this.clear = function() {
    this.dataStore = [];
    this.listSize = 0;
    this.pos = 0;
  }
  this.contains = function(el) {
    for(let i=0; i<this.dataStore.length; i++) {
      if(this.dataStore[i] === el) return true;
      return false;
    }
  };
  this.front = function() {
    this.pos = 0;
  };
  this.end = function() {
    this.pos = this.listSize -1;
  };
  this.prev = function() {
    if(this.pos > 0) --this.pos;
  };
  this.next = function() {
  if(this.pos < this.listSize-1) ++this.pos
  };
  this.currPos = function() {
    return this.pos;
  };
  this.moveTo = function(position) {
    this.pos = position
  };
  this.getElement = function() {
  return this.dataStore[this.pos]
  };

}
```

리스트와 반복

반복자를 이용해 List 클래스의 내부 저장소를 직접 참조하지 않고 리스트를 탐색하 수 있다.

List 클래스의 fron, end, prev, next, currPos 함수를 이용해 반복자를 구현할 수 있다. 반복자는 배열의 인덱스에 비해 다음과 같은 장점이 있다.

- 리스트 요소에 접근시 내부 데이터 저장소가 무엇인지 걱정할 필요가 없다.
- 리스트에 새 요소를 추가했을 때는 현재 인덱스가 쓸모없는 값이 되는 반면 반복자를 이용하면 리스트가 바뀌어도 반복자를 갱신할 필요는 없다.
- 리스트 클래스에 사용하는 데이터 저장소의 종류가 달라져도 이전과 같은 방식으로 요소에 접근할 수 있다.

```
for(names.front(); names.currPos() < names.length(); names.next()) {
  print(names.getElement())
}
```

자바스크립트의 리스트

비교적 최근에 등장한 언어의 창시자들은 리스트의 중요함을 알고 있었습니다. 이들은 리스트를 기본적으로 지원해 개발자들의 고충을 덜어주자는 생각을 했습니다.

자바스크립트의 경우는 배열에 리스트의 기능을 포함하고 있습니다.

아래는 인덱스 3인 엘리먼트를 삭제하는 코드입니다.

```
const nums = [10,20,30,40,50];
nums.splice(3,1);
for(let i=0; i<nums.length; i++) {
  console.log(nums[i]);
}
// 10
// 20
// 30
// 50
```

위 코드에서 중요한 부분은 아래입니다.

```
nums.splice(3,1);
```

spllice는 배열 3번째 인덱스 요소를 삭제합니다. 삭제된 엘리먼트가 있던 자리는 뒤의 요소에 의해 채워집니다.

즉, 자바스크립트의 배열은 리스트의 기능도 포함하고 있습니다. 자바스크립트 배열은 배열이자 리스트가 될 수 있다는 것을 의미합니다.

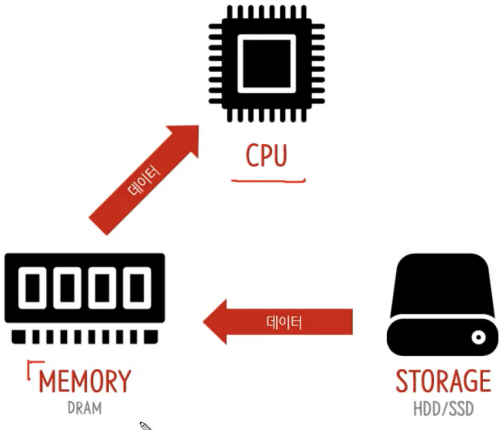
만약 각각의 요소가 고유한 인덱스를 가지고 있게 하려면 splice대신에 nums[3] = null; 과 같은 방법을 사용하면 됩니다.

즉, **배열을 어떻게 이용하느냐에 따라서 배열이 될 수도 있고 리스트도 될 수 있습니다**.

Linked List

KEY → 연결

메모리



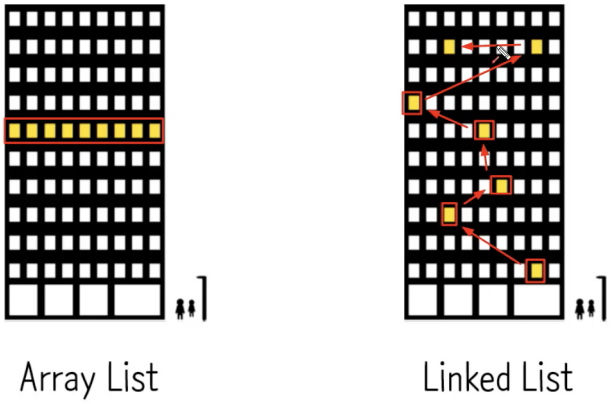
메모리(RAM)	**스토리지**
가격비쌈	가격저렴
용량이 적음	용량이 큼
전원이 꺼지면 데이터 사라짐	전원이 꺼져도 데이터 저장되어 있음
빠름	느림

이런 이유로 데이터는 기본적으로 스토리지에 저장됩니다. 하지만 스토리지는 매우 느리기 때문에 CPU와 함께 일을 하기에는 속도면에서 부족합니다. 그래서 어떤 프로그램을 실행하면 그 프로그램과 데이터는 메모리로 옮겨집니다. CPU는 메모리에 로드된 데이터를 이용해 여러 가지 일을 합니다.

그러므로 실행 속도를 결정하는 것은 대체로 메모리입니다. 우리가 데이터 스트럭처를 배우는 이유는 메모리의 효율적인 사용이라고 할 수 있습니다.

Array List vs Linked List

Array	Linked List
인덱스	연결
연속된 메모리 공간에 존재	메모리 상에서 떨어져 있는 데이터들이 앞의 데이터가 뒤를 데이터를 기억하는 형개로 존재
데이터 조회시: O(1)	데이터 조회시: O(N)
데이터추가 및 삭제: O(N)	데이터추가 및 삭제: O(1)



Array List

첫 번째 회사는 모든 직원이 한 곳에 모여 있어야 한다는 철학이 있기 때문에 한층을 임대해 사무실이 모여 있습니다. 배열은 건물을 이런 식으로 사용하는 것과 비슷합니다. 만약 회사가 성장해서 사무실이 좁아 진다면 새로운 직원을 뽑을 수 없습니다. 붙어 있는 공간이 없기 때문이죠. 만약 더 많은 공간이 필요하다면 더 많은 사람을 수용할 수 있는 공간을 찾아 전체가 이사가야 합니다.

Linked List

linked list는 한 건물내에서 한 최사가 임대한 사무실이 서로 떨어져 있습니다. 덕분에 직원이 늘어도 큰 걱정이 없습니다. 건물에서 비어있는 곳을 임대해서 들어가면됩니다. 그런데 방문자가 사무실을 찾는 방법이 좀 비효율적입니다. 3번째 사무실을 찾아 가야 한다면 첫 번째 화살표부터 사무실찾아 물어물어 이동해야 합니다. 이것이 linked list입니다. 그래서 linked list 에서 몇 번째 엘리먼트를 찾는 것은 느립니다.

반면에 array list 는 엘리먼트가 한 곳에 모여있습니다. 찾고자 하는 사무실이 몇번째에 있는지 알고 있다면 array list는 매우 빠릅니다.

연결

배열과 달리 linked list는 그 위치가 흩어져 있기 때문에 서로 연결되어 있어야합니다. 바로 그런 점에서 연결이라는 이름을 갖게 된 것입니다.

linked list는 다양한 데이터 스트럭쳐에서 광범위하게 사용되는 개념이기 때문에 잘 이햐하셔야 합니다.

데이터스트럭처의 본질. 즉, 각각의 데이터 스트럭처의 중심되는 컨셉을 알아야 하는 것이 중요!! 데이터 스트럭처의 본질을 알지 못하면 어떤 경우에 무엇을 쓸지 베스트 프랙티스를 전부 외워야함! 무리!!!!