

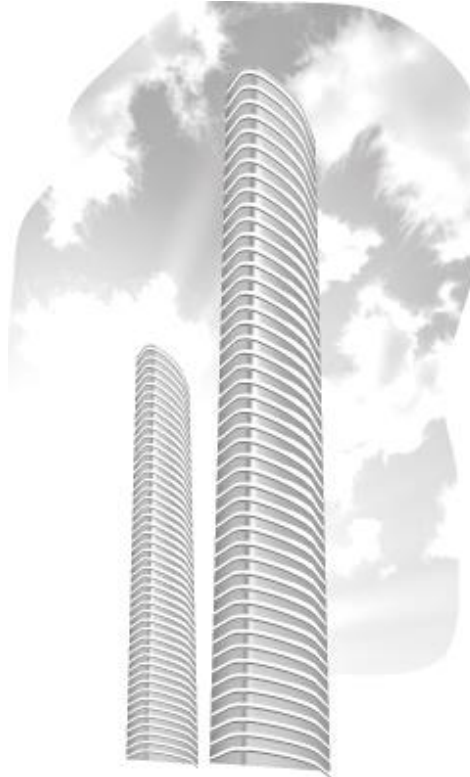
# 자료구조의 시작

# 목차

1. 시작하기 전에
2. 자료구조의 정의
3. 자료구조의 분류
4. 추상 자료형
5. 알고리즘

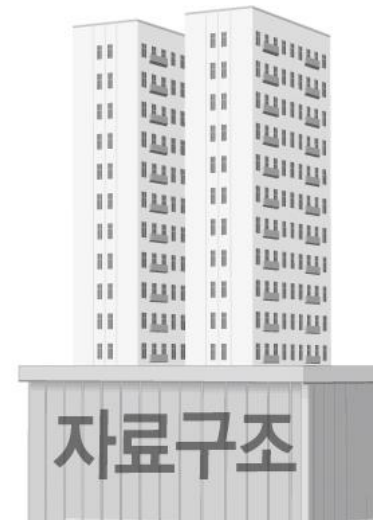
# 시작하기 전에 (1/2)

- 자료구조(Data Structure)의 필요성
  - (비유) 프로그램 개발 ≍ 건물 건축
    - 장난감 집 VS 고층 아파트



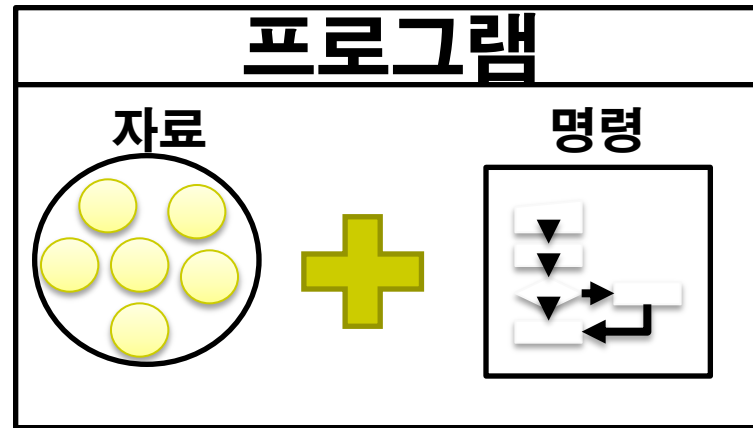
## 시작하기 전에 (2/2)

- 자료구조의 역할
  - 컴퓨터 프로그래밍에 있어서 가장 기초적인 학문분야
  - 컴퓨터 프로그램의 기본 골격
    - 프로그램이 효율적이고 안전하게 동작하게 하기 위해서 반드시 필요
- 간과하기 쉬운 경우
  - 프로그램의 크기가 작은 경우
  - 대형 프로젝트의 초기 단계
    - 구조적인 결함 발생



# 1. 자료 구조의 정의 (1)

- 프로그램의 구조
  - 자료(데이터,data)와 명령으로 구성



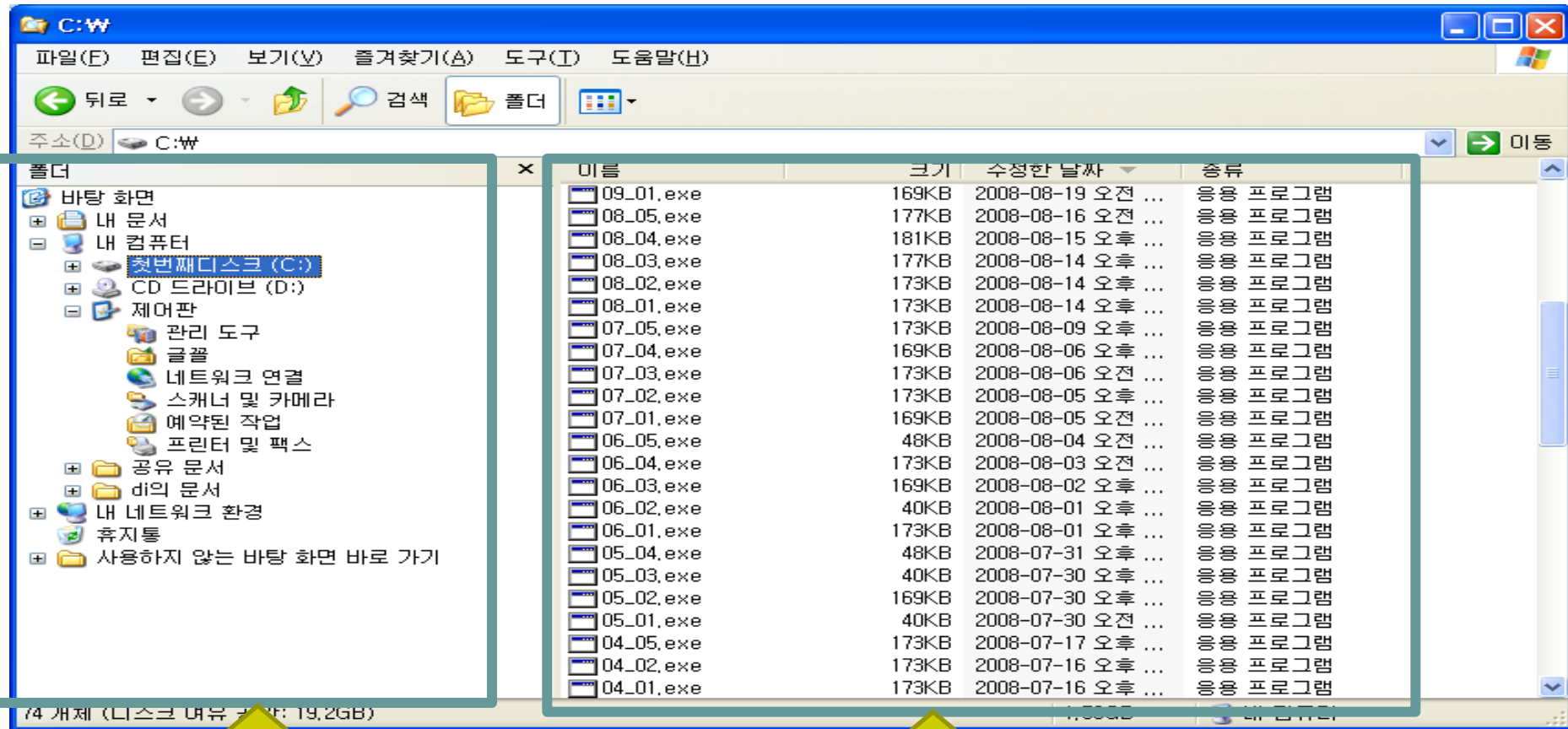
예	프로그램의 공통점	
	명령	자료
윈도우 탐색기	파일의 복사, 이동 및 삭제	파일 및 폴더의 계층 구조 정보
전자사전	단어 검색	단어의 철자 및 의미

# 1. 자료 구조의 정의 (2)

- 자료구조

- 컴퓨터에 자료를 효율적으로 저장하는 방식
  - 메모리(저장 공간) 절약 (Space Complexity)
  - 프로그램 수행(실행) 시간 단축 (Time Complexity)
- 프로그램의 수행 시간 혹은 저장 공간을 고려한 자료구조의 설계
  - ⇒ 프로그램이 어떻게 사용되는지에 따라 결정  
(프로그램의 목적 및 기능에 부합하는 자료구조 설계)
- 예
  - 윈도우 탐색기
  - 사전 프로그램

## 예) 윈도우 탐색기

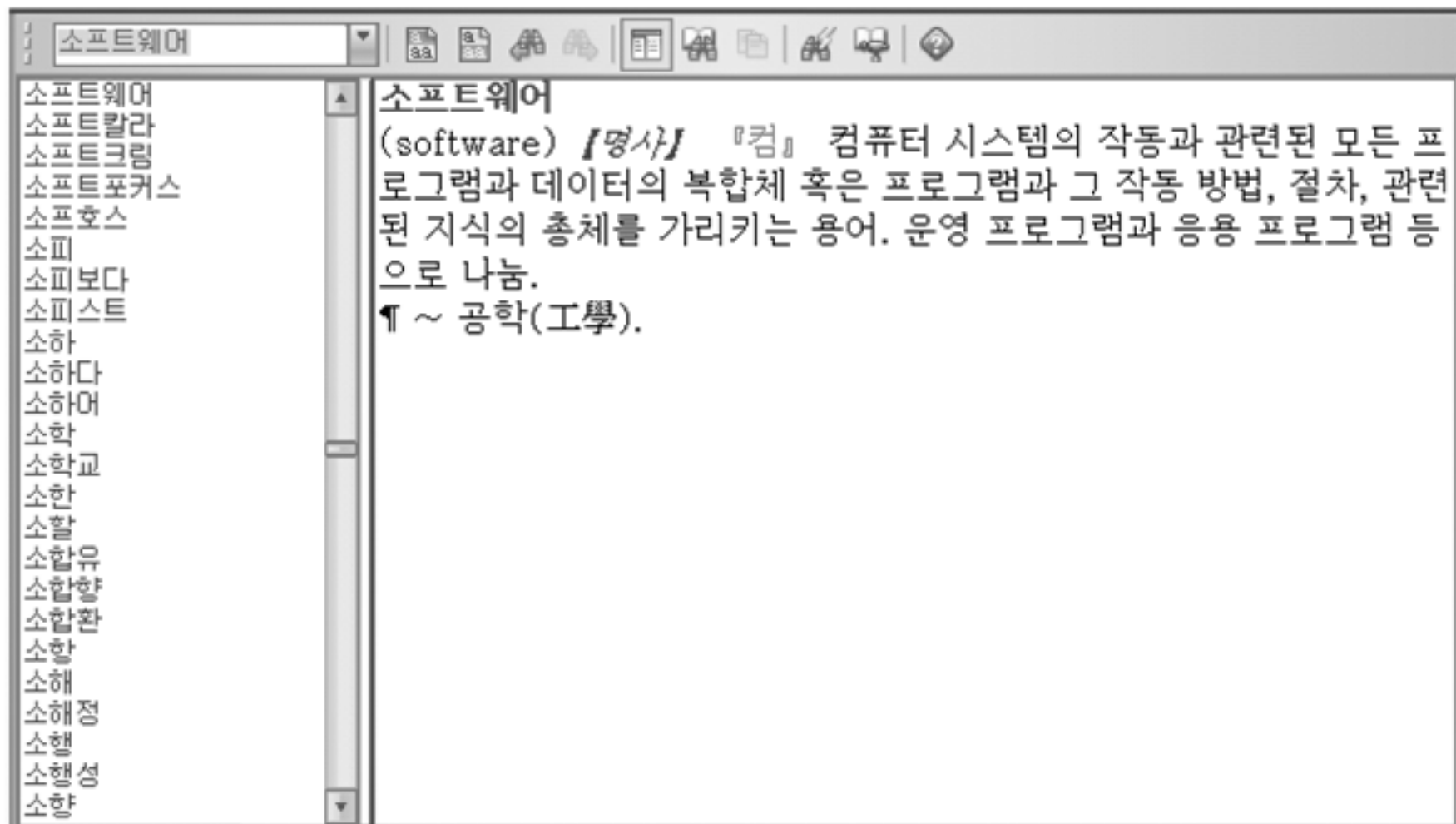


(a) 폴더 구조 7장 트리

(b) 파일 목록 3장 리스트

## 예) 사전 프로그램

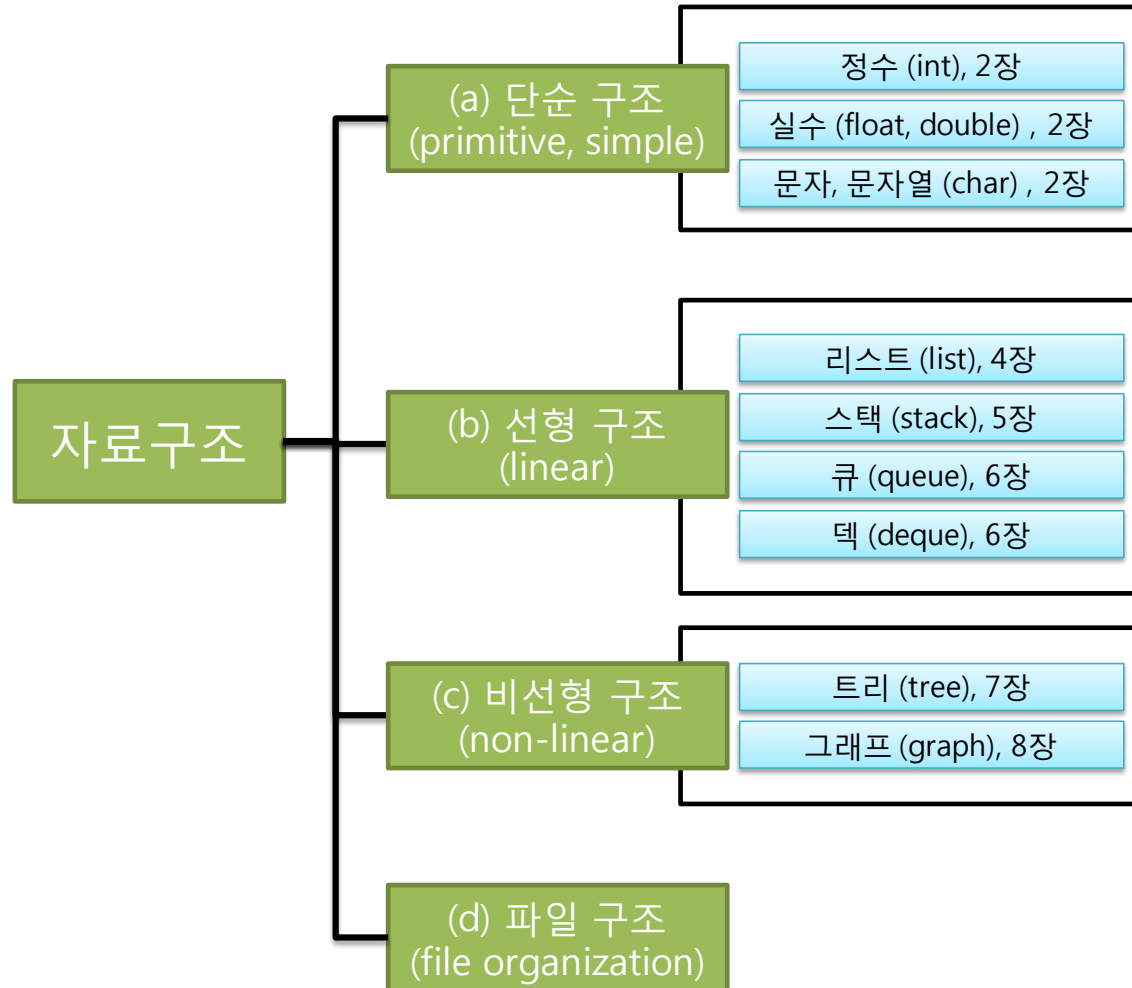
### 10장 검색





## 2. 자료 구조의 분류 (1)

- 자료(data)의 형태에 따른 분류



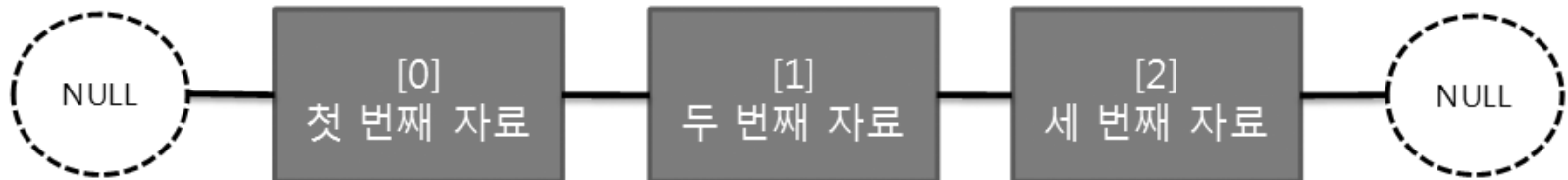
## 2. 자료 구조의 분류 (2)

### (1) 단순 구조

- 프로그래밍 언어에서 제공하는 기본적인 데이터 타입
- 예) 정수(int), 실수(float 혹은 double), 문자와 문자열(char) 등

### (2) 선형 구조

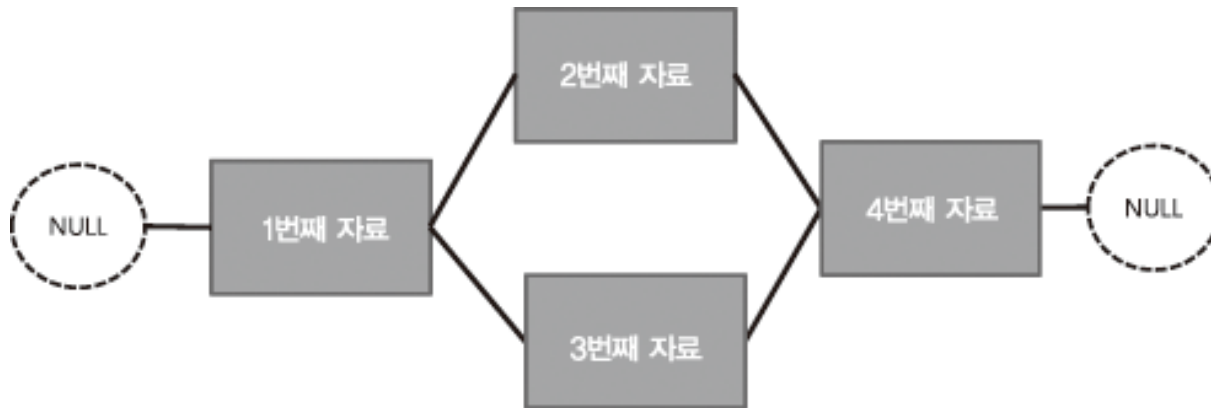
- 자료들 사이의 앞뒤 관계가 일대일(1:1)인 경우
- 예) 3장의 리스트(List), 4장의 스택(Stack), 5장의 큐(Queue)와 덱(Deque) 등



## 2. 자료 구조의 분류 (3)

### (3) 비선형 구조

- 자료들 사이의 앞뒤 관계가 계층 구조(Hierarchical Structure) 혹은 망 구조(Network Structure)를 가지는 경우
- 예) 7장 트리(Tree). 8장(그래프) 등



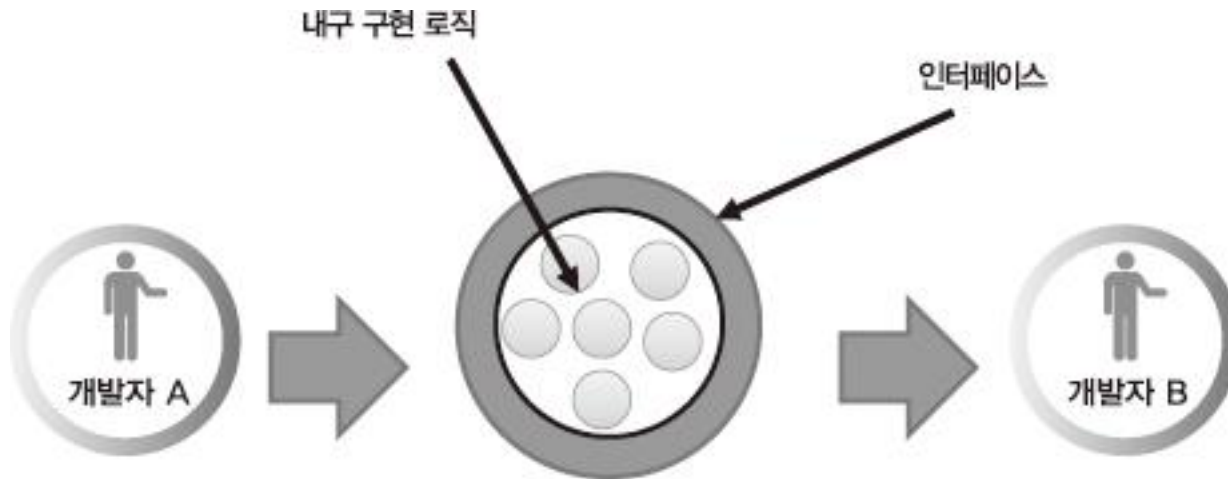
### (4) 파일 구조

- 보조기억장치(예를 들어 하드디스크)에 저장되는 파일에 대한 자료구조

\* (2),(3) 컴퓨터 메모리상에서 저장된다는 가정

### 3. 추상 자료형 (1)

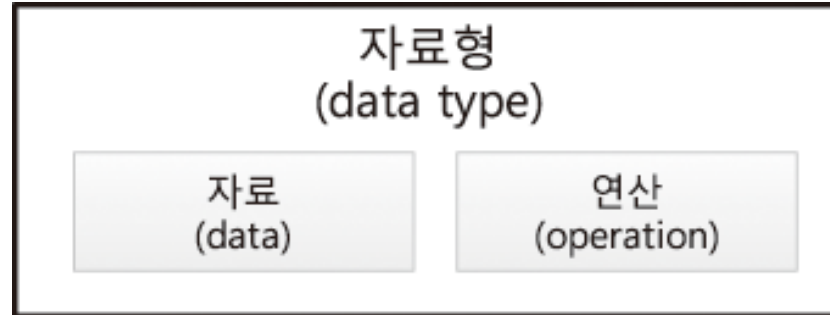
- 추상 자료형(Abstract Data Type, ADT)
  - 자료구조를 기술(표현)하는 가장 대표적인 방법 중 하나
  - 정보 은닉(Information Hiding)
    - 중요한 정보만을 나타내고, 중요하지 않은 정보는 숨기는 것



- 자료구조를 사용할 수 있는 인터페이스(Interface) 정의
  - 혹은 자료구조의 연산(Operation)들의 정의

### 3. 추상 자료형 (2)

- 자료형(Data Type)



- 예) 정수 자료형

구분		예
정수 자료형 (int data type)	자료 (data)	..., -2, -1, 0, 1, 2, ...
	명령 (operation)	더하기 (+)
		빼기 (-)
		곱하기 (*)
		나누기 (/)
		나머지 연산 (%)

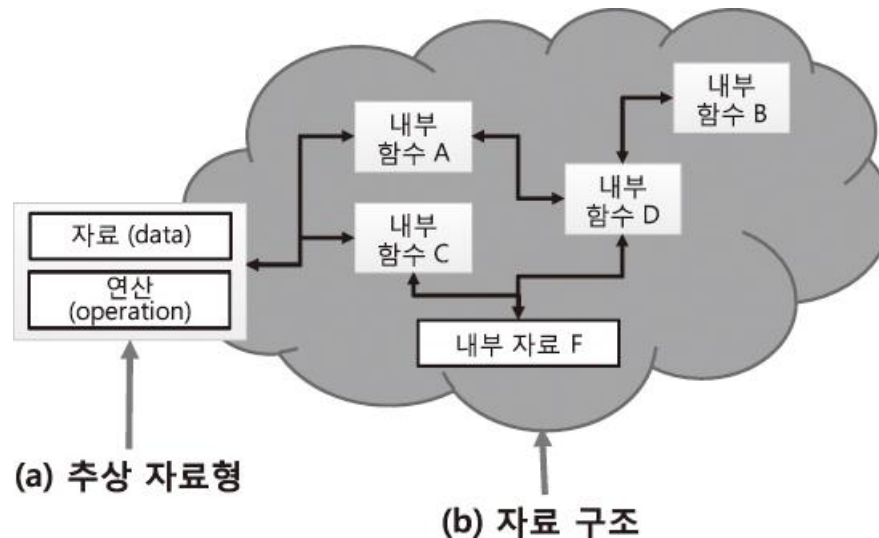
### 3. 추상 자료형 (3)

- 추상 자료형
  - 추상적으로 정의된 자료형
  - 추상(Abstraction)의 의미
    - 1) 세부적이고 복잡한 것을 생략
    - 2) 대표적인 것, 중요한 것만을 나타냄
  - 예) 추상 자료형 MyPosition
    - 1. 자료(2개)
      - 1) X 좌표의 위치를 나타내는 정수
      - 2) Y 좌표의 위치를 나타내는 정수
    - 2. 명령:

이름		입력(Input)	출력(Output)	설명
1) 변경	modify()	1. 위치 자료 p 2. X축 변경 거리: x 3. Y축 변경 거리: y	N/A (없음)	위치 p에서 X축으로 x만큼, Y축으로 y만큼 변경
2) 위치 사이 거리 계산	distance()	1. 위치 자료 p1 2. 위치 자료 p2	두 위치 사이의 거리	두 위치 p1, p2 사이의 거리를 구한다
3) 위치 비교	equal()	1. 위치 자료 p1 2. 위치 자료 p2	TRUE 혹은 FALSE	두 위치 p1, p2가 같은 위치인지 조사

### 3. 추상 자료형 (4)

- 자료구조를 이용하는 사용자의 관점
  - 자료구조의 내부 구현 소스에 대한 분석 없이 신속하게 자료구조를 이용 가능
  - 예) 이름, 입력, 출력
- 자료구조를 공급하는 개발자 관점
  - 자료구조를 구현하기 전에 설계도를 미리 그려보는 것
  - 자료구조의 개발자와 사용자 사이의 간섭 문제가 해결



## 4. 알고리즘 (1)

- 알고리즘(Algorithm)
  - 넓은 의미: 자료구조와 함께 컴퓨터 프로그램을 구성하는 한가지 요소
    - 컴퓨터 명령어들의 집합
  - 좁은 의미: 어떠한 문제를 해결하기 위한 절차
  - 예
    - 1부터 100까지 합을 구하는 문제

$$\begin{array}{r} 1 \\ + 2 \\ + 3 \\ \dots\dots \\ + 100 \\ \hline 5,050 \end{array}$$

### 의사 코드(Pseudo Code)

```
CalcSum( n ) {  
    sum ← 0  
    for( i ← 1 ; i ≤ n ; i ← i + 1 ) {  
        sum ← sum + i;  
    }  
    return sum;  
}
```



## 4. 알고리즘 (2)



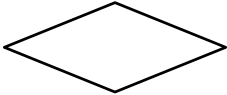
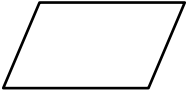

- 알고리즘의 필수 5가지 특성
  - 1) 입력(input): 외부에서 제공되는 자료가 0개 이상 있어야 한다.
  - 2) 출력(output): 적어도 1개 이상의 결과를 만들어야 한다.
  - 3) 명백성(definiteness): 각 명령어는 의미가 모호하지 않고 명확해야 한다.
  - 4) 유한성(finiteness): 한정된 수의 단계 뒤에는 반드시 종료된다. 무한히 동작해서는 안 된다.
  - 5) 유효성(effectiveness): 모든 명령은 실행 가능한 연산이어야 한다. 예를 들어 0으로 나누는 연산과 같이 실행할 수 없는 연산을 포함해서는 안 된다.
- 알고리즘과 자료구조와의 관계
  - 알고리즘의 성능은 자료구조에 종속
    - 같은 자료라 하더라도 어떻게 표현되고 저장되느냐에 따라 사용 가능한 알고리즘이 달라지기 때문
  - 자료구조의 기본적인 연산을 구현하기 위해서 알고리즘이 사용

## 4.1. 알고리즘의 표현

종류	내용	특성
자연어 (Natural Language)	사람이 사용하는 일반적인 언어로 표현	알고리즘 표현으로 부적절 : 기술하는 사람에 따라 일관성과 명확성이 달라짐
순서도 (Flow Chart)	알고리즘을 그림으로 도식화해서 표현	장점: 알고리즘 각 단계를 직관적으로 표현 단점: 복잡한 알고리즘을 나타내기에는 비효율적
의사 코드 (Pseudo Code)	특정 프로그래밍 언어가 아닌 가상의(유사한) 언어로 표현	프로그래밍 언어보다는 덜 엄격한 문법이나, 자연어보다는 더 체계적으로 기술이 가능 자료구조/알고리즘 책마다 약간씩 구문에 차이가 있다. 책에 따라 가상코드, 유사 코드 혹은 슈도 코드라 불린다.
프로그래밍 언어 (Programming Language)	C와 같은 프로그래밍 언어로 표현	장점: 추가 구현 단계가 필요 없다 (알고리즘을 구현 소스를 통해 나타내기 때문) 단점: 너무 엄격하게 기술하여 비효율적인 경우가 있다.

## 4.2 순서도와 의사 코드 (1)

- 순서도

기호	내용
	시작(start) 혹은 종료(end)
	처리(process): 특정 연산을 실행한다.
	판단(decision): 주어진 조건을 비교하여 해당되는 조건에 따라 흐름을 결정한다.
	자료(data): 자료의 입력(input) 혹은 출력(output)을 나타낸다
	흐름선(flow line): 실행 순서를 표시한다.

## 4.2 순서도와 의사 코드 (2)

- 의사 코드(Pseudo Code)

- (1) 지정문

- 변수에 특정 값을 대입하는 명령

문법	예제	순서도
변수 $\leftarrow$ 값 ;	sum $\leftarrow$ sum + 10;	<div>sum <math>\leftarrow</math> sum + 10</div>

## 4.2 순서도와 의사 코드 (3)

- 의사 코드 (계속)

- (2) 조건문

- 조건식을 만족하는지에 따라 흐름이 결정되는 명령

문법	예제	순서도
<pre>if (조건식1) then {     명령문1 ; } else if (조건식2) then {     명령문2 ; } else {     명령문3 ; }</pre>	<pre>if (sum &gt;= 90) then {     grade ← 'A' } else if (sum &gt;= 80) then {     grade ← 'B' } else {     grade ← 'C' }</pre>	<pre>graph TD     D1{sum &gt;= 90} -- Yes --&gt; P1[grade ← 'A']     D1 -- No --&gt; D2{sum &gt;= 80}     D2 -- Yes --&gt; P2[grade ← 'B']     D2 -- No --&gt; P3[grade ← 'C']</pre>

## 4.2 순서도와 의사 코드 (4)

- 의사 코드 (계속)

### (3) 반복문

- 특정 명령을 여러 번 반복적으로 수행하는 구조, for 또는 while

문법	예제	순서도
<pre>for(초기값; 조건식; 증감값) {     명령문 ; }</pre>	<pre>sum ← 0; for(i ← 1; i ≤ 100; i ← i+1) {     sum ← sum + i; }</pre>	<pre>graph TD     A[sum ← 0] --&gt; B[i ← 1]     B --&gt; C{i ≤ 100}     C -- Yes --&gt; D[sum ← sum + i]     D --&gt; E[i ← i + 1]     E --&gt; C     C -- No --&gt; F[ ]</pre>

## 4.3 알고리즘의 성능 분석 (1)

- 주어진 문제를 해결하기 위해 여러 개의 다양한 알고리즘이 가능  $\Rightarrow$  어떤 알고리즘을 사용해야 하는가?
  - 알고리즘의 성능 분석 필요
  - 예: 1부터 100까지 합을 구하는 문제

(a) 100번의 연산	(a) 3번의 연산
$\begin{array}{r} 1 \\ + 2 \\ + 3 \\ + 4 \\ \dots \\ + 100 \\ \hline 5,050 \end{array}$	$\frac{100 \times (1 + 100)}{2} = 5,050$
100번의 연산 (덧셈 100번)	3번의 연산 (덧셈 1번, 곱셈 1번, 나눗셈 1번)

## 4.3 알고리즘의 성능 분석 (2)

- 알고리즘 분석 기준
  - 공간 복잡도(Space Complexity)
  - 시간 복잡도(Time Complexity)
    - 실제 걸리는 시간을 측정
    - 실행되는 명령문의 개수를 계산
      - 시간 복잡도 함수: 입력 값에 따른 실행 연산의 빈도수

**입력 값: n**

(a) 알고리즘 1	(a) 알고리즘 2
<pre>CalcSum( n ) {   i ← start; // 1번   sum ← 0; // 1번   for( ; i ≤ n; i ← i + 1 ) { // 1+1번     sum ← sum + i; // 1 번   }   return sum; }</pre>	<pre>CalcSum( n ) {   count ← n;   sum ← 1 + n;   sum ← sum * count;   sum ← sum / 2   return sum; }</pre>

**알고리즘 1의 연산 빈도수:**  $2 + n * 3 = 3n + 2$

**알고리즘 2의 연산 빈도수:** 4



## 4.3 알고리즘의 성능 분석 (3)

- 시간 복잡도  $\asymp$  빅-오( $O$ ) 표기법
  - 빅-오 표기법(Big-Oh Notation)
  - 시간 복잡도 함수 중에서 가장 큰 영향력을 주는  $n$ 에 대한 항만을 표시
  - 계수(Coefficient)는 생략하여 표시
  - 예

$$O(3n + 2) = \frac{O(3n)}{\text{최고차항}(3n)\text{만 선택}} = \frac{O(n)}{\text{계수 3제거}}$$

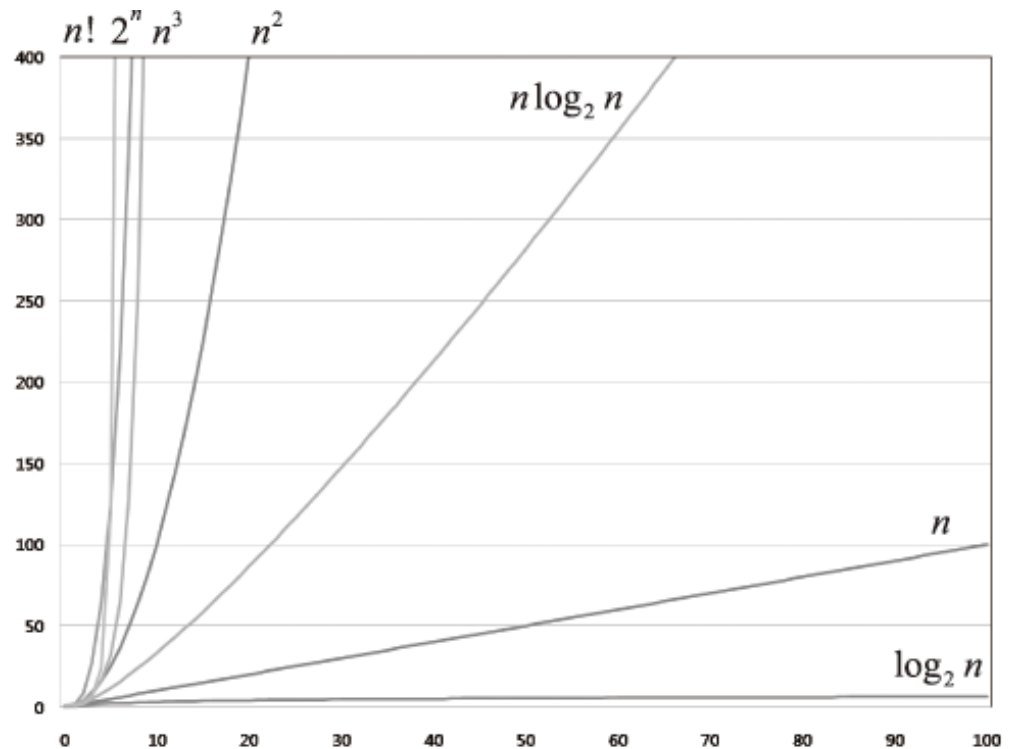
$$O(4) = O(1)$$

- 정확한 정의

두 개의 함수  $f(n)$ 과  $g(n)$ 이 주어졌을 때, 모든  $n \geq n_0$ 에 대해  $|f(n)| \leq c|g(n)|$ 을 만족하는 2개의 상수  $c$ 와  $n_0$ 가 존재하면  $f(n) = O(g(n))$ 이다.

## 4.3 알고리즘의 성능 분석 (4)

- 시간 복잡도에서 많이 사용되는 최고 차항들
  - $O(1)$ : 상수
  - $O(\log n)$ : 로그
  - $O(n)$ : 1차, 선형
  - $O(n \log n)$ : 선형로그
  - $O(n^2)$ : 2차
  - $O(n^3)$ : 3차
  - $O(2^n)$ : 지수(Exponential)
  - $O(n!)$ : 계승(Factorial)



# 이번 장에서는

- 시작하기 전에
  - 자료구조의 필요성과 역할
- 자료구조의 정의
  - 컴퓨터에 자료를 효율적으로 저장하는 방식
- 자료구조의 분류
  - 단순구조 / 선형구조 / 비선형 구조 / 파일 구조
- 추상 자료형
  - 추상의 개념과 정보 은닉
- 알고리즘
  - 알고리즘의 좁은 의미에서의 정의 및 필수 특성
  - 표현: 순서도와 의사 코드
  - 성능 분석