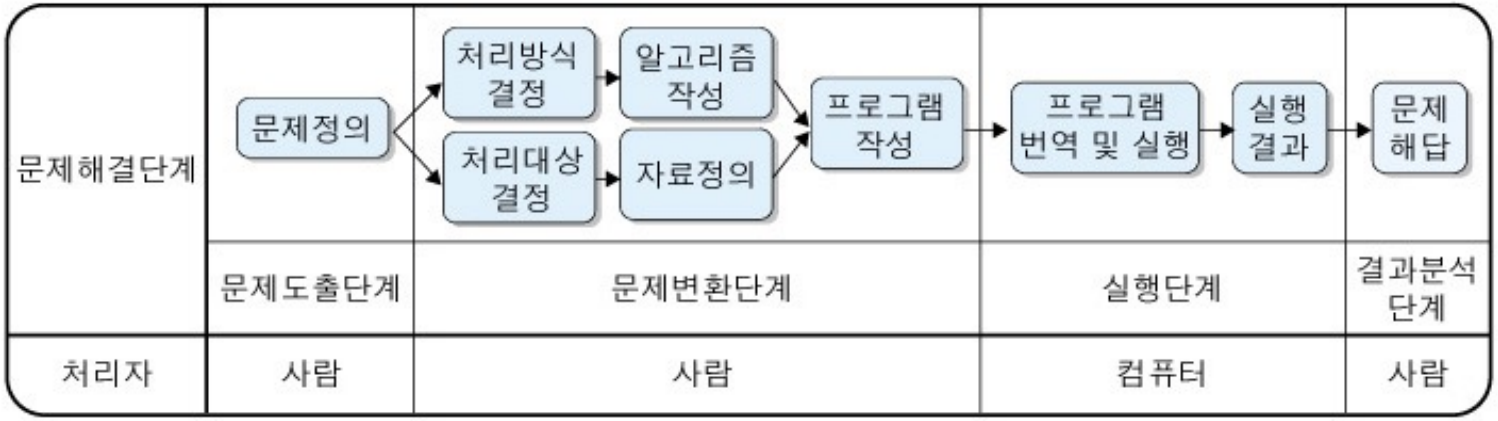
개요

❖ 컴퓨터에 의한 문제 해결 과정



❖ 자료구조(Data Structure)

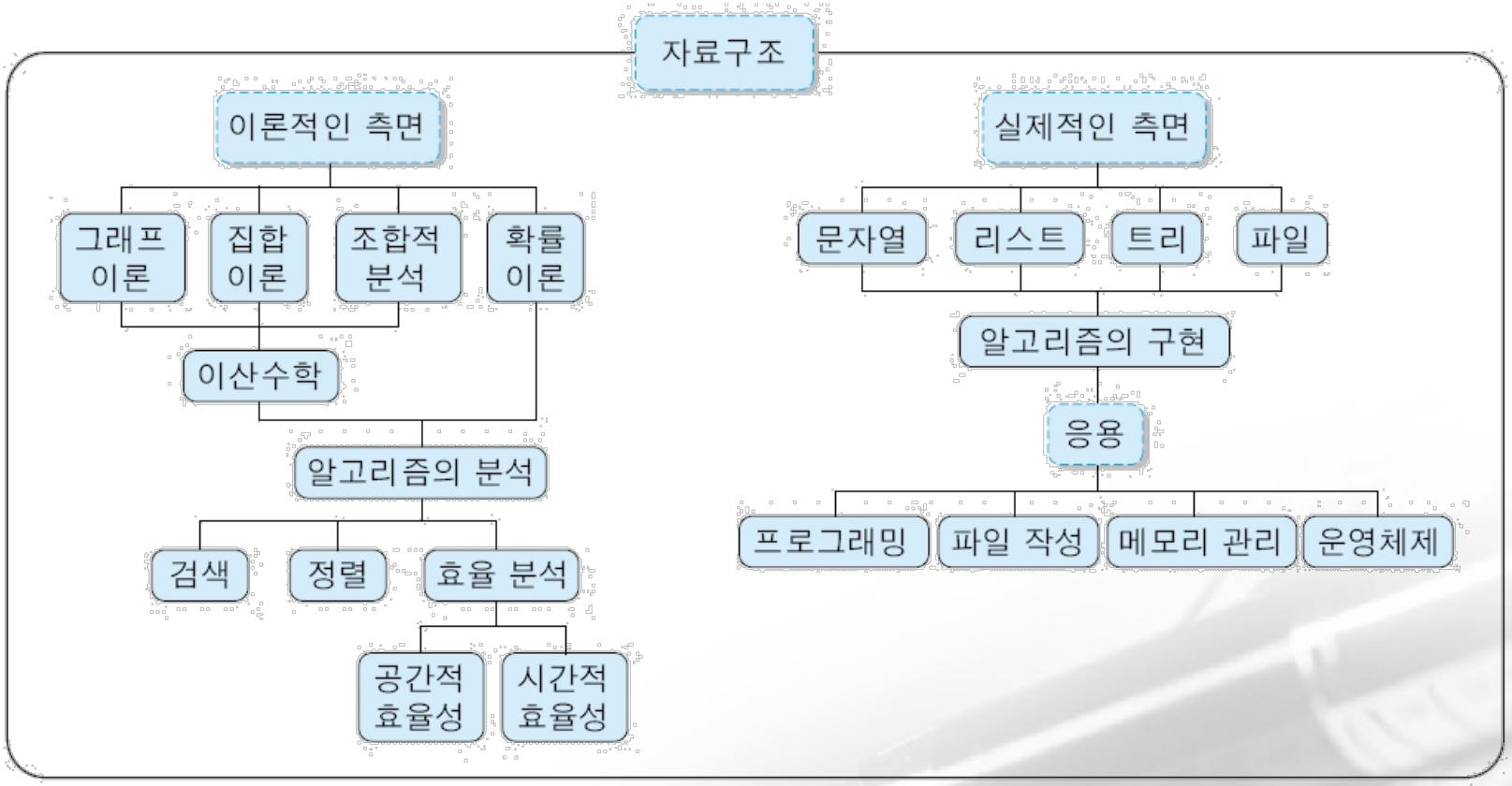
- ✓ 컴퓨터에 자료를 저장하는 방식
- ✓ 자료를 효율적으로 사용하기 위해서 자료의 특성에 따라서 분류하고 구성한 후 저장 및 처리하는 모든 작업
- ✓ 효율적인 자료구조는 메모리를 절약할 뿐 아니라 프로그램 수행 시간을 최소화하는 기능을 수행

개요

- ❖ 자료구조(Data Structure) 분류
  - ✓ 단순 구조 – Scala Data
    - ○ 데이터 1개를 저장하기 위한 구조
    - ○ 정수, 실수, 문자, 문자열 등의 기본 자료형
  - ✓ 선형구조
    - ○ 자료들 간의 앞뒤 관계가 1:1의 선형 관계
    - ○ Dense List (배열, 연접 리스트), Linked List, Stack, Queue, Deque 등
  - ✓ 비선형구조
    - ▪ 자료들 간의 앞뒤 관계가 1:N, 또는 N:N의 관계
    - ▪ Tree, Graph 등
  - ✓ 파일구조
    - ▪ 레코드의 집합인 파일에 대한 구조
    - ▪ Sequential File, Indexed File, Direct File 등

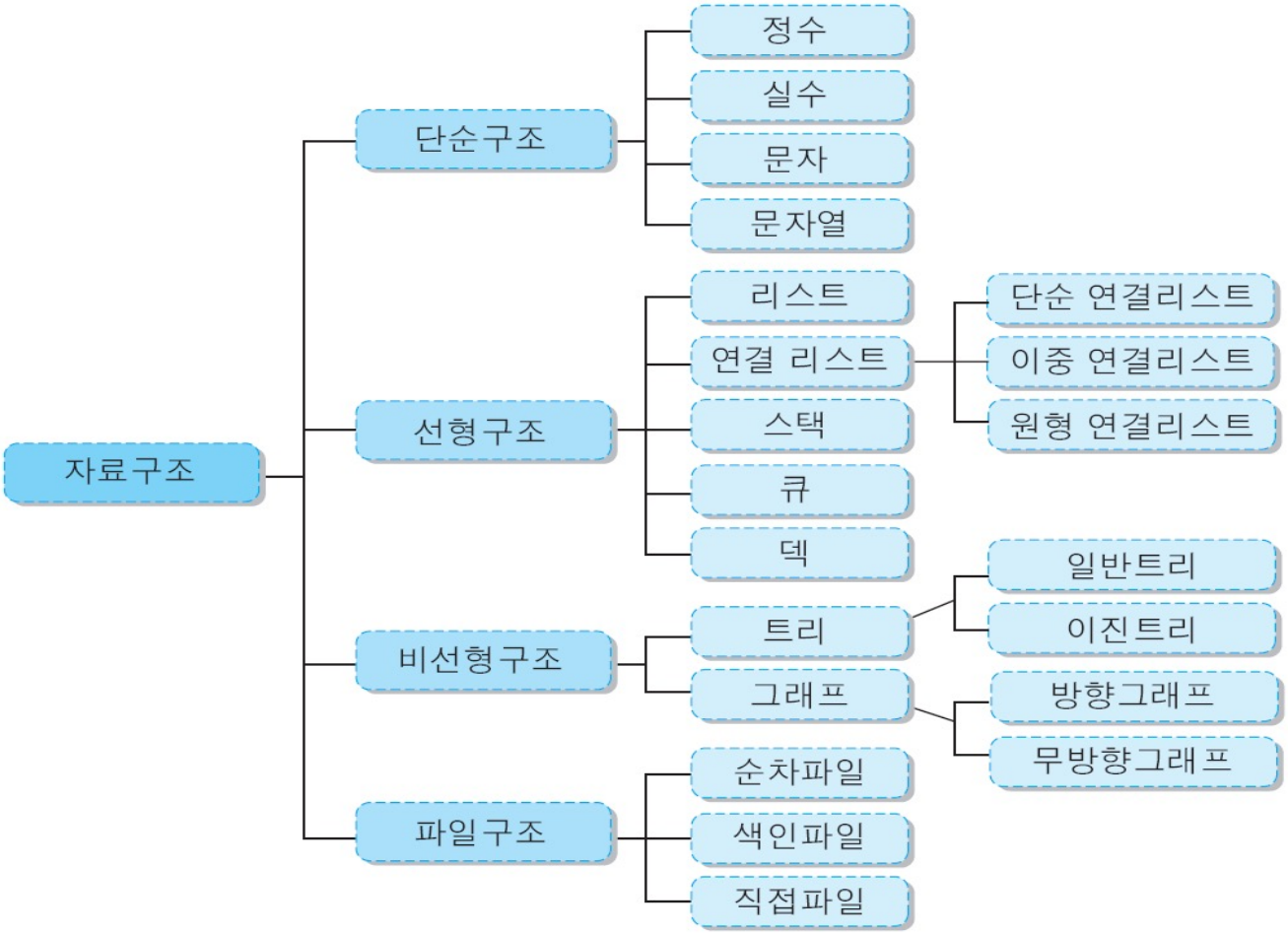
개요

❖ 자료구조(Data Structure) 분류



개요

❖ 자료구조(Data Structure) 분류



개요

- 알고리즘
  - ✓ 어떠한 문제를 해결하기 위한 절차
  - ✓ 문제 해결 방법을 추상화하여 각 절차를 논리적으로 기술해 놓은 명세서
  - ✓ 알고리즘의 조건
    - 입력(input): 알고리즘 수행에 필요한 자료가 외부에서 입력으로 제공될 수 있어야 함
    - 출력(output): 알고리즘 수행 후 하나 이상의 결과를 출력해야 함
    - 명확성(definiteness): 수행할 작업의 내용과 순서를 나타내는 알고리즘의 명령어들은 명확하게 명세되어야 함
    - 유한성(finiteness): 알고리즘은 수행 뒤에 반드시 종료되어야 함
    - 효과성(effectiveness): 알고리즘의 모든 명령어들은 기본적인 명령이어야 하며 실행이 가능해야 함
  - ✓ 알고리즘과 자료구조의 관계
    - 같은 자료라 하더라도 어떻게 표현되고 저장되느 냐에 따라 사용 가능한 알고리즘이 달라짐
    - 효율적인 자료구조의 설계는 알고리즘의 설계에 영향을 끼치기 때문에 프로그램의 성능을 결정짓게 하는 가장 중요한 요소 중 하나
    - 효율적으로 설계되지 않은 자료구조는 알고리즘을 비효율적으로 만들 뿐 아니라 문제 해결을 위한 알고리즘 자체를 만들어 내기 어려움

개요

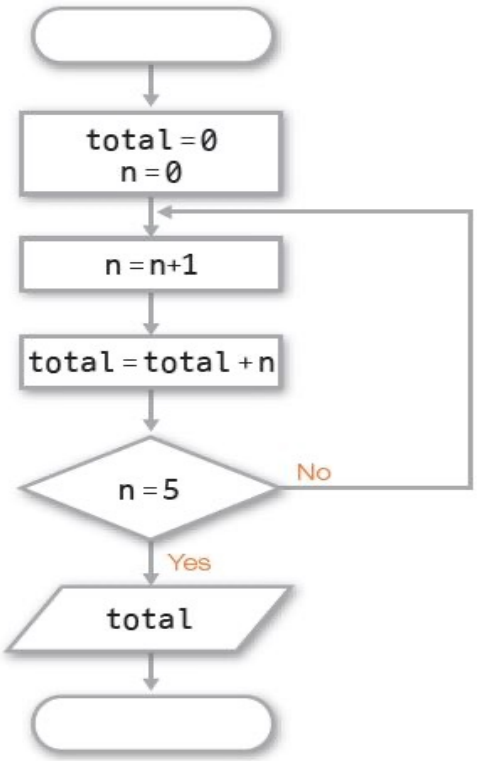
- ❖ 알고리즘
  - ✓ 알고리즘의 표현
    - ○ 자연어를 이용한 서술적 표현 방법
      - ▪ 기술하는 사람에 따라 일관성과 명확성이 달라지기 때문에 알고리즘 표현으로 부적절
    - ○ 순서도(Flow chart)를 이용한 도식화 표현 방법
      - ▪ 알고리즘 각 단계를 직관적으로 표현할 수 있는 장점이 있으나 복잡한 알고리즘을 나타내기에는 비효율적
    - ○ 가상 코드(Pseudo-code)를 이용한 추상화 방법
      - ▪ 프로그래밍 언어보다는 덜 엄격한 문법이나 자연어보다는 더 체계적으로 기술이 가능
      - ▪ 표현하는 개발자마다 약간씩 구문에 차이가 있음
    - ○ 프로그래밍 언어를 이용한 구체화 방법
      - ▪ 구체적인 구현 소스를 통해 나타내기 때문에 추가 구현 단계가 필요없음
      - ▪ 너무 엄격하게 기술하기 때문에 비효율적인 경우가 많음

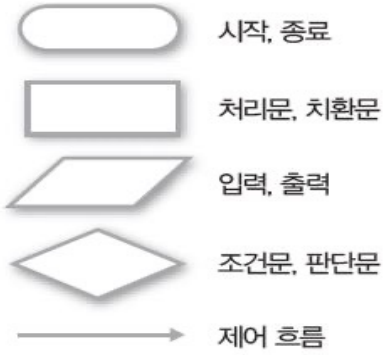
개요

❖ 알고리즘 개요

✓ 순서도를 이용한 알고리즘 표현

1부터 5까지의 합을 구하는 알고리즘





개요

- ❖ 알고리즘 개요
  - ✓ 가상 코드(Pseudo-Code)를 이용한 알고리즘 표현
    - ○ 가상 코드 즉 알고리즘 기술 언어(ADL, Algorithm Description Language)를 사용하여 프로그래밍 언어의 일반적인 형태와 유사하게 알고리즘을 표현
    - ○ 특정 프로그래밍 언어가 아니므로 직접 실행은 불가능
    - ○ 일반적인 프로그래밍 언어의 형태이므로 원하는 특정 프로그래밍 언어로의 변환 용이
    - ○ 기본 요소
      - ▪ 기호
        - ❑ 변수, 자료형 이름, 프로그램 이름, 레코드 필드 명, 문장의 레이블 등을 나타냄
        - ❑ 문자나 숫자의 조합. 첫 문자는 반드시 영문자 사용
      - ▪ 자료형
        - ❑ 정수형과 실수형의 수치 자료형, 문자형, 논리형, 포인터, 문자열 등의 모든 자료형 사용
      - ▪ 연산자
        - ❑ 산술연산자, 관계연산자, 논리연산자

개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- 대입

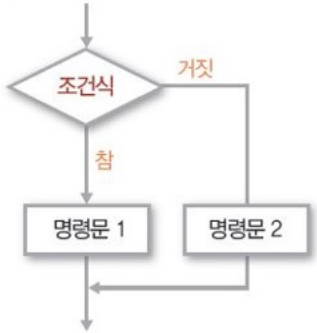
```
변수 ← 값
```

```
a ← 5
a ← 3+2
a ← b;
```

- 조건에 따른 분기

```
if (조건식) then 명령문 1;
```


```
if (조건식) then 명령문 1;
else 명령문 2;
```



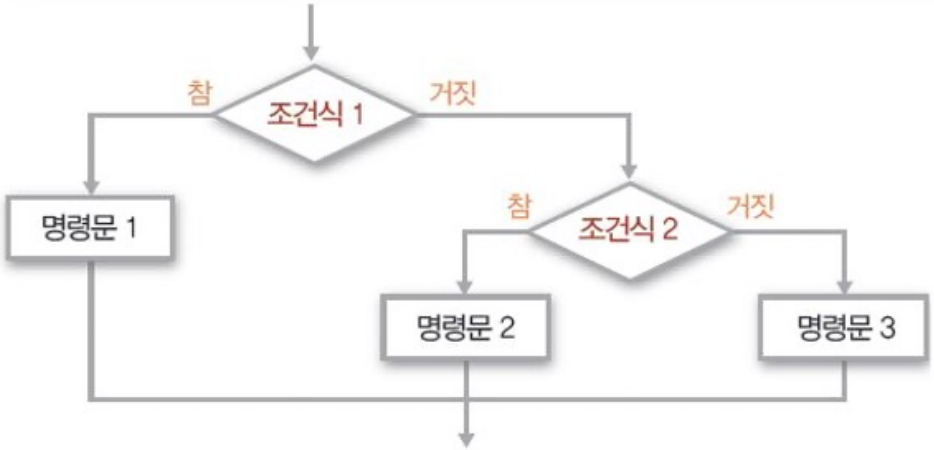
개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- 조건에 따른 분기

```
if (조건식 1) then 명령문 1;
else if (조건식 2) then 명령문 2;
else 명령문 3;
```



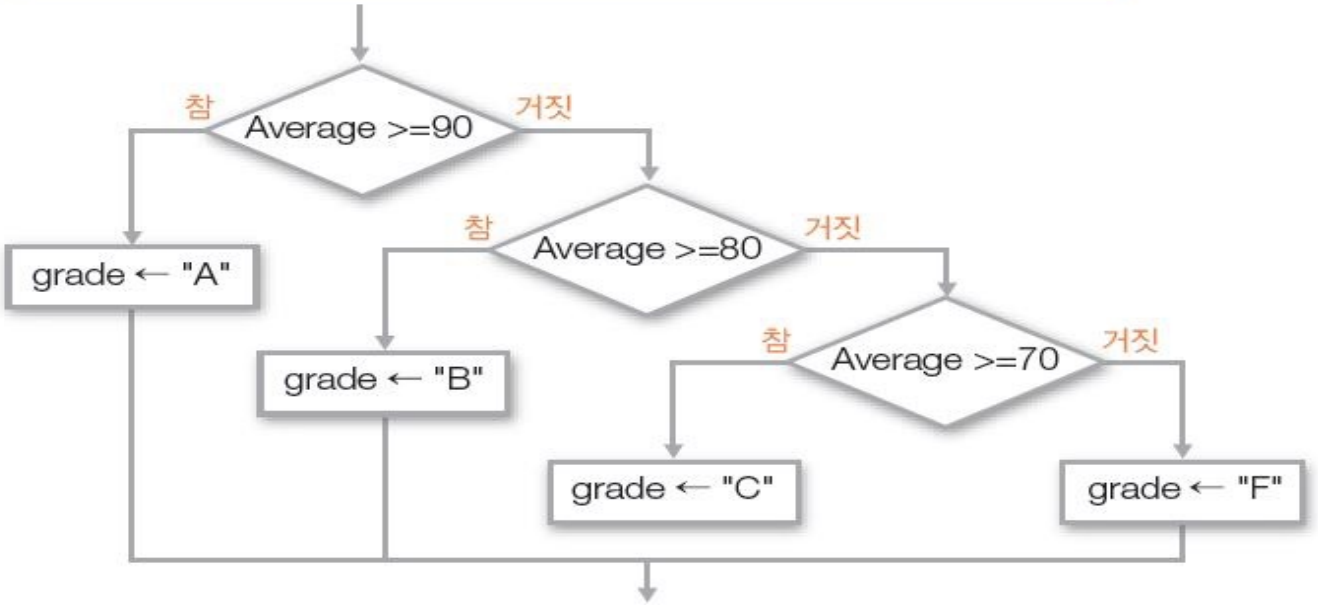
개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- 조건에 따른 분기

```
if Average >= 90 then grade ← "A";
else if Average >= 80 then grade ← "B";
else if Average >= 70 then grade ← "C";
else grade ← "F";
```



개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- 값에 따른 분기

```
case {
    조건식 1 : 명령문 1;
    조건식 2 : 명령문 2;
    ...
    조건식 n : 명령문 n;
    else    : 명령문 n+1;
}
```

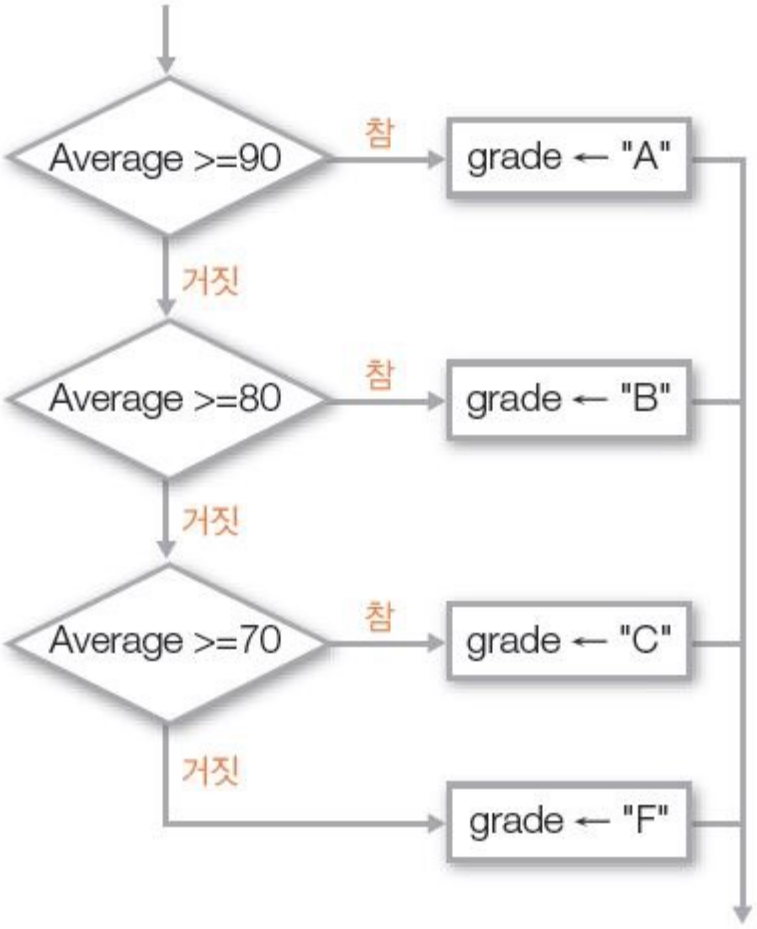
개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- 값에 따른 분기

```
case {
    Average >= 90 : grade ← "A";
    Average >= 80 : grade ← "B";
    Average >= 70 : grade ← "C";
    else :          grade ← "F";
}
```

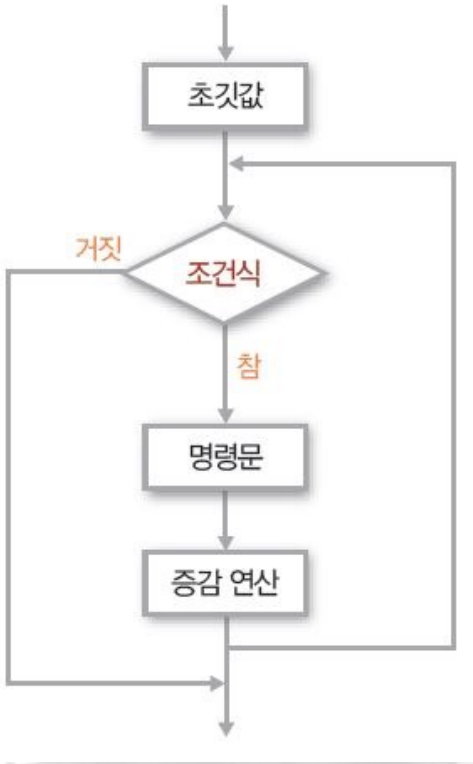


개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- for를 이용한 반복




```
for (초깃값; 조건식; 증감값) do 명령문;
```

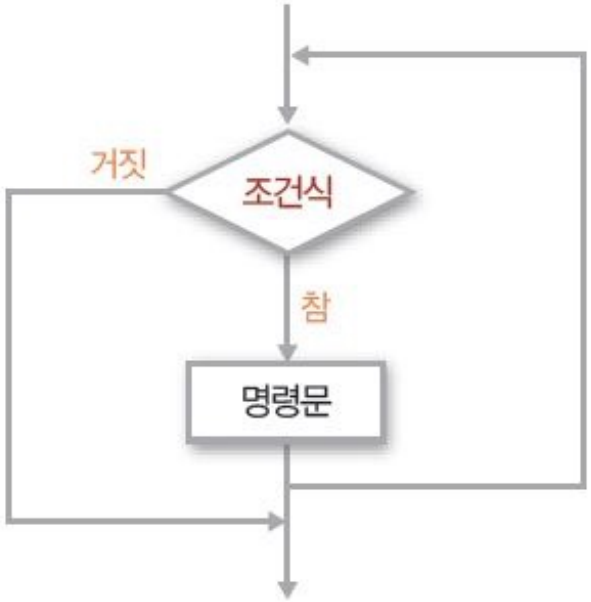
개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- while를 이용한 반복

```
while (조건식) do 명령문;
```



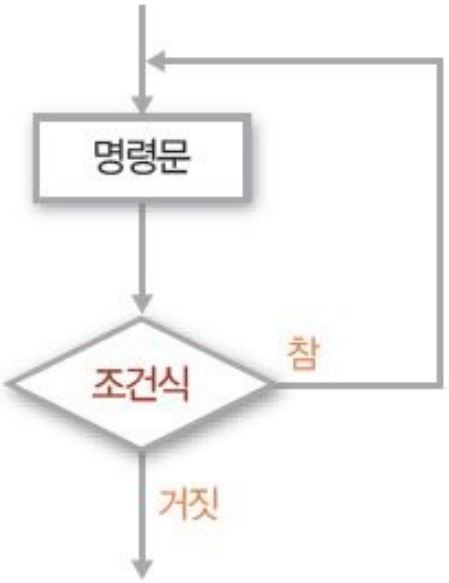
개요

❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

- do ~ while를 이용한 반복

```
do 명령문;
while (조건식);
```



개요

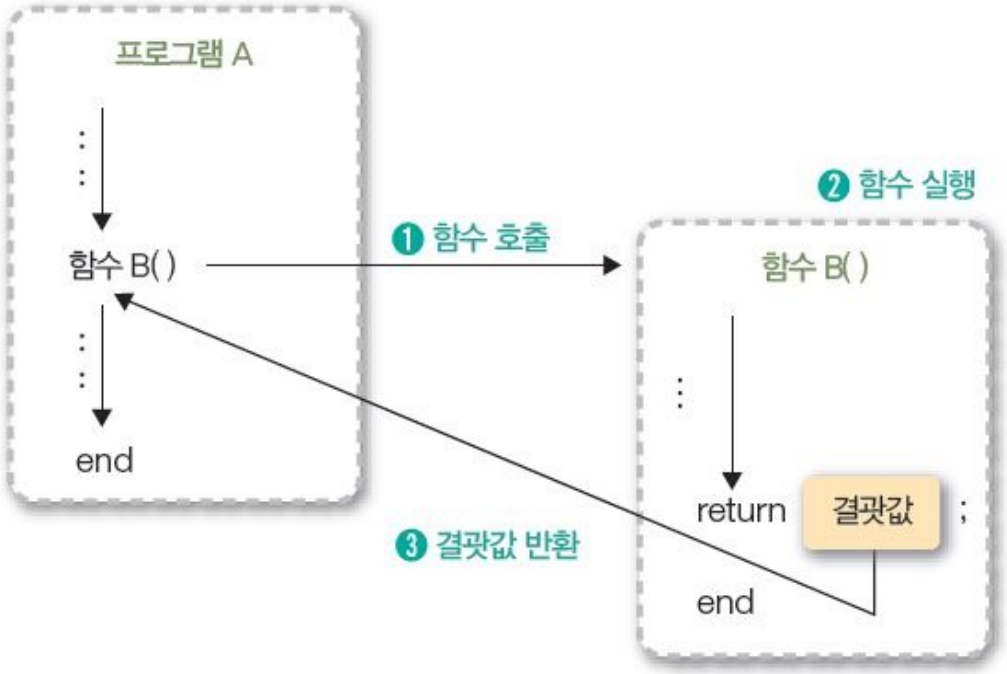
❖ 알고리즘 개요

✓ 가상코드(Pseudo-Code)를 이용한 알고리즘 표현

○ 함수 호출

```
함수 이름 (매개변수)
    명령문;
    ...
    return 결괏값;
end
```

(a) 함수의 형식



(b) 함수의 호출과 실행 및 결괏값 반환의 예

개요

- ❖ 알고리즘 분석
  - ✓ 알고리즘 성능 분석 기준
    - 기준에는 정확성, 명확성, 수행량, 메모리 사용량, 최적성 등 있음
    - 정확성 : 올바른 자료 입력 시 유한한 시간 내에 올바른 결과 출력 여부
    - 명확성 : 알고리즘이 얼마나 이해하기 쉽고 명확하게 작성되었는가
    - 수행량 : 일반적인 연산 제외, 알고리즘 특성 나타내는 중요 연산 모두 분석
    - 메모리 사용량
    - 최적성

개요

- ❖ 알고리즘 분석
  - ✓ 알고리즘 성능 분석 방법
    - ○ 공간 복잡도
      - ▪ 알고리즘을 프로그램으로 실행하여 완료하기까지 필요한 총 저장 공간의 양
      - ▪ 공간 복잡도 = 고정 공간 + 가변 공간
    - ○ 시간 복잡도
      - ▪ 알고리즘을 프로그램으로 실행하여 완료하기까지의 총 소요시간
      - ▪ 시간 복잡도 = 컴파일 시간 + 실행 시간
      - ▪ 컴파일 시간 : 프로그램마다 거의 고정적인 시간 소요
      - ▪ 실행 시간 : 컴퓨터의 성능에 따라 달라질 수 있으므로 실제 실행시간 보다는 명령문의 실행 빈도수에 따라 계산
      - ▪ 실행 빈도수의 계산
      - ▪ 지정문, 조건문, 반복문 내의 제어문과 반환문은 실행시간 차이가 거의 없으므로 하나의 단위시간을 갖는 기본 명령문으로 취급
    - ○ 알고리즘의 성능 분석에 시간 복잡도가 공간 복잡도보다 더 중요한 평가 기준이기 때문에 알고리즘의 성능 분석은 대부분 시간 복잡도를 대상으로 함

개요

❖ 알고리즘 분석

✓ 시간 복잡도

피보나치 수열

```
00  fibonacci(n)
01      if (n<0) then
02          stop;
03      if (n≤1) then
04      return n;
05      f1 ← 0;
06      f2 ← 1;
07      for (i ← 2; i≤n; i ← i + 1) do {
08          fn ← f1 + f2;
09          f1 ← f2;
10          f2 ← fn;
11      }
12      return fn;
13  end
```

개요

❖ 알고리즘 분석

✓ 시간 복잡도

○ 입력 값에 따른 실행 연산의 빈도수

피보나치 수열 알고리즘의 실행 빈도수

(a) n<0, n=0, n=1인 경우

행 번호	n<0	n=0	n=1
01	1	1	1
02	1	0	0
03	0	1	1
04	0	1	1
05~13	0	0	0

(b) n>1 경우

행 번호	n>1	행 번호	n>1
01	1	08	n-1
02	0	09	n-1
03	1	10	n-1
04	0	11	0
05	1	12	1
06	1	13	0
07	n		

개요

- ❖ 알고리즘 분석
  - ✓ 시간 복잡도
    - ○ 빅-오(O) 표기법
      - O(f(n))과 같이 표기, "Big Oh of f(n)"으로 읽음
      - 수학적 정의: 함수 f(n)과 g(n)이 주어졌을 때, 모든 n ≥ n0에 대하여 |f(n)| ≤ c |g(n)| 을 만족하는 상수 c와 n0이 존재하면 f(n) = O(g(n))
      - 함수의 상한을 나타내기 위한 표기법: 최악의 경우에도 g(n)의 수행 시간 안에는 알고리즘 수행 완료 보장
      - 먼저 실행 빈도수를 구하여 실행 시간 함수를 찾고, 이 함수 값에 가장 큰 영향을 주는 n에 대한 항을 한 개 선택하여 계수는 생략하고 O의 오른쪽 괄호 안에 표시
      - 피보나치 수열에서 실행 시간 함수는 4n+2이고, 가장 영향이 큰 항은 4n인데 여기에서 계수 4를 생략하여 O(n)으로 표기

개요

- ❖ 알고리즘 분석
  - ✓ 시간 복잡도
    - ○ 빅-오메가 표기법
      - Ω(f(n))과 같이 표기, "Big Omega of f (n)"으로 읽음
      - 수학적 정의: 함수 f(n)과 g(n)이 주어졌을 때, 모든 n≥n0에 대하여 |f(n)| ≥ c |g(n)|을 만족하는 상수 c와 n0이 존재하면, f(n) = Ω(g(n))
      - 함수의 하한을 나타내기 위한 표기법: 어떤 알고리즘의 시간 복잡도가 Ω(g(n))으로 분석되었다면, 이 알고리즘 수행에는 적어도 g(n)의 수행 시간이 필요함을 의미

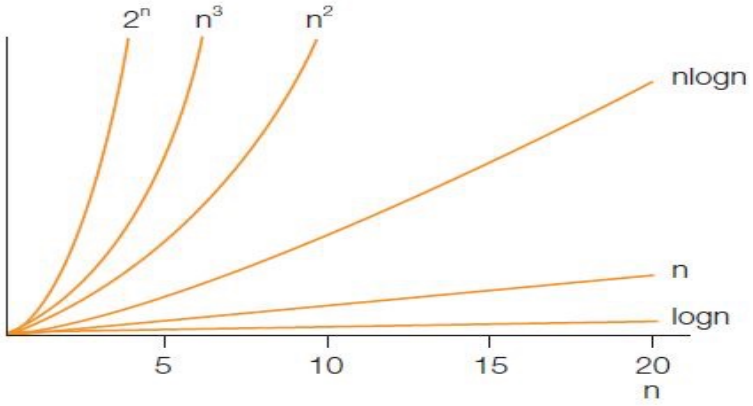
개요

- ❖ 알고리즘 분석
  - ✓ 시간 복잡도
    - ○ 빅-세타 표기법
      - θ(f(n))과 같이 표기, "Big Theta of f (n)"으로 읽음
      - 수학적 정의: f(n)과 g(n)이 주어졌을 때, 모든 n≥n0에 대하여 c1|g(n)| ≤ f(n) ≤ c2 |g(n)| 을 만족하는 상수 c1, c2와 n0이 존재하면, f(n) = θ(g(n))
      - 상한과 하한이 같은 정확한 차수를 표현하기 위한 표기법: f(n)= θ (g(n))이 되려면 f (n)= O(g(n))이면서 f (n)= Ω(g(n))이어야 함

개요

- ❖ 알고리즘 분석
  - ✓ 시간 복잡도
    - ○ 각 실행 시간 함수에서 n값의 변화에 따른 실행 빈도수 비교

logn <	n <	nlogn <	n^2 <	n^3 <	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296



개요

❖ 알고리즘 분석

✓ 시간 복잡도

○ 시간 복잡도에 따른 알고리즘 수행 시간 비교

입력 크기 n	알고리즘 수행 시간				
	n	nlogn	n^2	n^3	2^n
10	10^{-8}초	3×10^{-8}초	10^{-7}초	10^{-6}초	10^{-6}초
30	3×10^{-8}초	2×10^{-7}초	9×10^{-7}초	3×10^{-5}초	1초
50	5×10^{-8}초	3×10^{-7}초	3×10^{-6}초	10^{-4}초	13일
100	10^{-7}초	7×10^{-7}초	10^{-5}초	10^{-3}초	4×10^{13}년
1,000	10^{-6}초	10^{-5}초	10^{-3}초	1초	3×10^{283}년
10,000	10^{-5}초	10^{-4}초	10^{-1}초	17분	
100,000	10^{-4}초	2×10^{-3}초	10초	12일	
1,000,000	10^{-3}초	2×10^{-2}초	17분	32년	