승자는 같은 결과를 얻기 위해
다른 방법을 사용하고,
패자는 같은 방법을 사용하여
다른 결과를 얻으려고 한다.

# 1장 알고리즘과 문제해결

# 목차

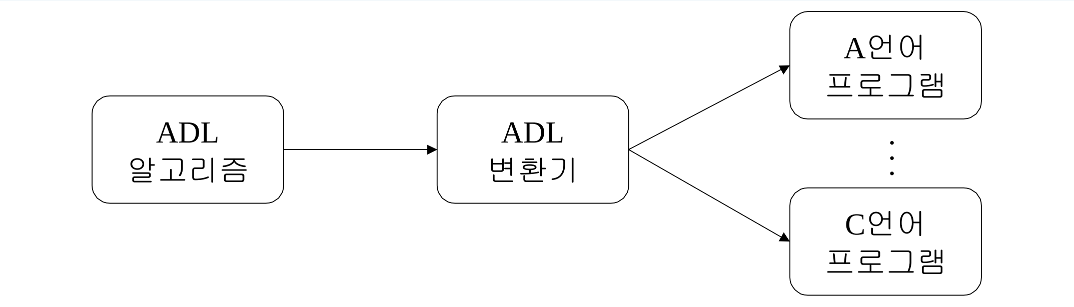
- 알고리즘이란?
- 알고리즘 기술 언어
- 알고리즘 성능 분석
  - 공간 복잡도
  - 시간 복잡도
- 순환과 점화 관계

# 알고리즘이란?

- 알고리즘(algorithm)
  - 특정문제를 해결하기 위해 기술한 일련의 명령문
- 프로그램(program)
  - 알고리즘을 컴퓨터가 이해하고 실행할 수 있는 특정 프로그래밍 언어로 표현한 것
- 알고리즘의 요건
  - 완전성과 명확성
    - 수행결과와 순서가 완전하고 명확하게 명세되어야 함
    - 순수하게 알고리즘이 지시하는대로 실행하기만 하면 의도한 결과가 얻어져야 함
  - 입력과 출력
    - 입력 : 알고리즘이 처리대야 할 대상으로 제공되는 데이타
    - 출력 : 입력 데이타를 처리하여 얻은 결과
  - 유한성
    - 유한한 단계 뒤에는 반드시 종료

# 알고리즘 기술언어

- ADL (Algorithm Description Language)
  - 알고리즘 기술을 위해 정의한 언어
  - 사람이 이해하기 쉽고, 프로그래밍 언어로의 변환이 용이
  - 의사 코드 (pseudo-code) : ADL과 약간의 자연어로 기술한 것
  - ADL 알고리즘에서 프로그램으로의 변환



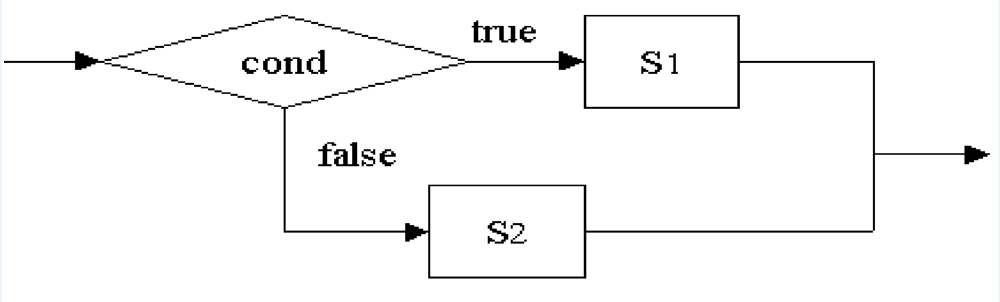

  - ADL 데이타 : 숫자, 부울(Boolean) 값, 문자
  - ADL의 명령문 :
    - 종류 : 지정문, 조건문, 반복문, 함수문, 입력문, 출력문
    - 명령문 끝에는 세미콜론(;)을 사용

# ADL – 지정문

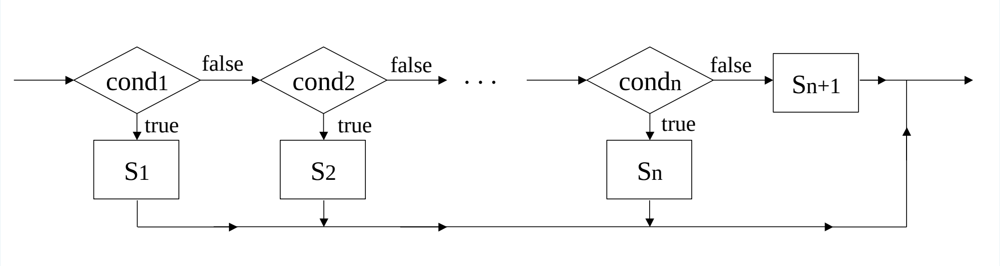
- 지정문
  - 형식 : 변수 ← 식;
  - 식 (expression)
    - 산술식
    - 부울식
      - 결과 : 참(true) 또는 거짓 (false)
      - 표현
        - 논리 연산자(and, or, not)
        - 관계 연산자(<, ≤, =, ≠, ≥, >)
    - 문자식
  - 제어 구조 : 순차적

# ADL – 조건문

- 조건문
  - 제어 구조 : 선택적
  - 종류 : if문과 case문
- if문
  - **if** (cond) **then** $S_1$
    **else** $S_2$



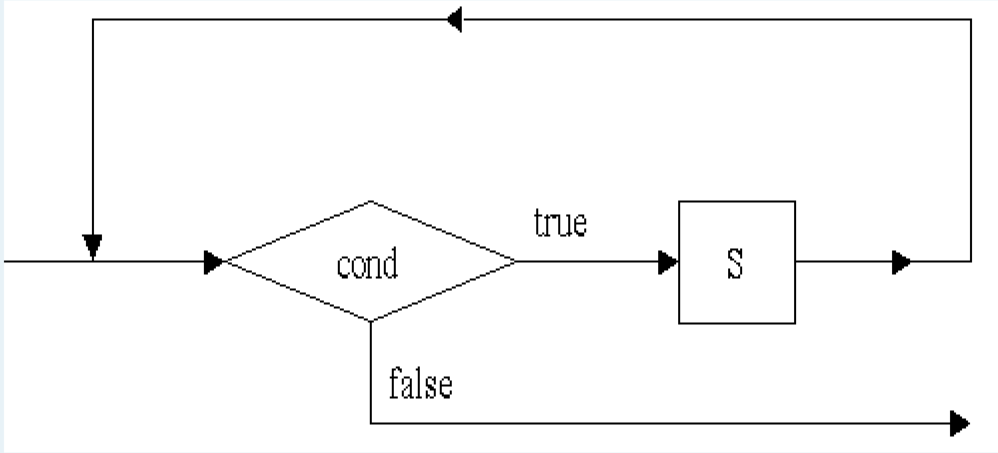

- case문
  - **case** {
    cond$_1$ : $S_1$
    cond$_2$ : $S_2$
    . . .
    cond$_n$ : $S_n$
    **else** : $S_{n+1}$
    }



# ADL – 반복문 (1)

- 반복문
  - 제어 구조 : 일정한 명령문 그룹을 반복해서 수행하는 루프(loop) 형태
  - 종류 : while문, for문, do-while문

- while문
  - 형식
    - **while** cond **do**
      S
  - 무한 루프
    - **while** true **do**
      S

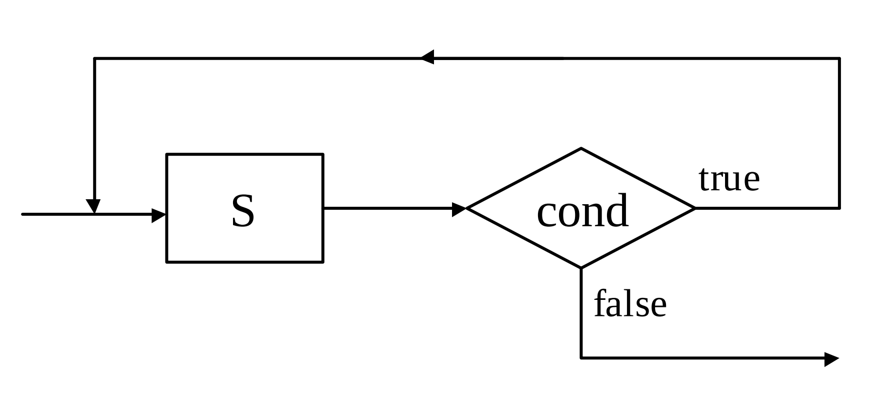


# ADL – 반복문 (2)

- for문
  - 형식
    - **for** (initialization; cond; increment) **do**
      S
  - 동등한 while문
    - initialization
      **while** cond **do** {
      S
      increment;
      }
  - 무한 루프
    - **for** ( ; ; ) **do**
      S
      (cond의 기정값이 true이므로)

# ADL – 반복문 (3)

- do-while문
  - 형식
    - do
      
      S
      
      while cond;




  - 특징
    - S가 최소한 한 번은 실행됨

- 루프 명령문
  - goto 명령문 : 루프에서 바로 빠져나갈 때 사용
  - exit문 : 자신을 둘러싸고 있는 가장 가까운 루프 밖의 명령문으로 제어를 이동시킴

# ADL – 함수문

- 함수문
  - 형식
    - function-name(parameter_list)
      
      S
      
      end
  - 호출 함수로의 복귀
    - **return** expr;
    - 여기서 expr은 함수의 실행 결과
  - 함수 호출
    - function-name(argument-list)
    - 인자 리스트(argument-list)는 타입과 수에 있어서 함수의 형식 매개 변수와 대응되어야 함
  - 인자와 매개변수와의 연관
    - 값 호출 (call by value) 규칙
    - 각 인자의 실제 값이 호출된 함수로 전달
    - 인자의 값이 주소(참조)가 되면 매개 변수에 주소 값이 전달되어 값은 데이타 지시

# ADL – 입출력문

- 입력 함수
  - read (argument_list);
- 출력 함수
  - print (argument_list);
- 인자
  - 변수나 인용 부호가 있는 문자열

# ADL – 기타

- 기타 명령문
  - 주석문 : //는 그 행 끝까지, /*과 */은 주석문의 시작과 끝 표시
  - 다차원 배열 : $a[n_1, n_2, \cdots , n_n]$

- ADL 기술 규칙
  - 함수의 입·출력 변수를 명확히 명세
  - 변수의 의미를 알 수 있게 정의
  - 알고리즘의 제어 흐름은 되도록 순차적
  - 시각적 구분을 위해 들여쓰기 이용
  - 코멘트는 짧으면서 의미는 명확히
  - 함수를 적절히 사용

# 성능 분석 (1)

- 알고리즘의 평가 기준
  - 원하는 결과의 생성 여부
  - 시스템 명세에 따른 올바른 실행 여부
  - 프로그램의 성능
  - 사용법과 작동법에 대한 설명 여부
  - 유지 보수의 용이성
  - 프로그램의 판독 용이

- 알고리즘의 성능 평가
  - 성능 분석 (performance analysis)
    - 프로그램을 실행하는데 필요한 시간과 공간의 추정
  - 성능 측정 (performance measurement)
    - 컴퓨터가 실제로 프로그램을 실행하는데 걸리는 시간 측정

# 성능 분석 (2)

- 공간 복잡도 (space complexity)
  - 알고리즘을 실행시켜 완료하는데 필요한 총 저장 공간
  - $S_a = S_c + S_e$
    - $S_c$ : 고정 공간
      - 명령어 공간, 단순 변수, 복합 데이타 구조와 변수, 상수
    - $S_e$ : 가변 공간
      - 크기가 변하는 데이타 구조와 변수들이 필요로 하는 저장 공간
      - 런타임 스택(runtime stack)을 위한 저장 공간

- 시간 복잡도 (time complexity)
  - 알고리즘을 실행시켜 완료하는데 걸리는 시간
  - $T_a = T_c + T_e$
    - $T_c$ : 컴파일 시간
    - $T_e$ : 실행 시간
      - 단위 명령문 하나를 실행하는데 걸리는 시간
      - 실행 빈도수 (frequency count)

# 점근식 표기법 (1)

- 점근식 표기법(Asymptotic notation)
  - Big-Oh (O)
  - Big-Omega (Ω)
  - Big-Theta (Θ)

- Big-Oh (O)
  - f, g가 양의 정수를 갖는 함수일 때, 두 양의 상수 a, b가 존재하고, 모든 $n \geq b$에 대해 $f(n) \leq a \cdot g(n)$ 이면, $f(n) = O(g(n))$
  - 예
    - $f(n) = 3n + 2$ : $f(n) = O(n)$ (a=4, b=2)
    - $f(n) = 1000n^2 + 100n - 6$ : $f(n) = O(n^2)$
    - $f(n) = 6 \cdot 2^n + n^2$ : $f(n) = O(2^n)$
    - $f(n) = 100$ : $f(n) = O(1)$

# 점근식 표기법 (2)

- 연산 그룹
  - 상수시간 : $O(1)$
  - 로그시간 : $O(\log n)$
  - 선형시간 : $O(n)$
  - n로그시간 : $O(n\log n)$
  - 평방시간 : $O(n^2)$
  - 입방시간 : $O(n^3)$
  - 지수시간 : $O(2^n)$
  - 계승시간 : $O(n!)$
- 연산 시간의 순서

  $$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

- $O(n^k)$ : polynomial time

# 점근식 표기법 (3)

- Big-Omega ($\Omega$)
  - f, g가 양의 정수를 갖는 함수일 때, 두 양의 상수 a, b가 존재하고, 모든 $n \geq b$에 대해 $f(n) \geq a \cdot g(n)$ 이면,

    $f(n) = \Omega(g(n))$
  - 예
    - $f(n) = 3n+2$ : $f(n) = \Omega(n)$ (a=3, b=1)
    - $f(n) = 1000n^2+100n-6$ : $f(n) = \Omega(n^2)$
    - $f(n) = 6 \cdot 2^n+n^2$ : $f(n) = \Omega(2^n)$

# 점근식 표기법 (4)

- Big-Theta($\Theta$)
  - f, g가 양의 정수를 갖는 함수일 때,
    세 양의 상수 a, b, c가 존재하고, 모든 $n \geq c$
    에 대해 $a \cdot g(n) \leq f(n) \leq b \cdot g(n)$ 이면,
    $f(n) = \Theta(g(n))$
  - 예
    - $f(n) = 3n+2$ : $f(n) = \Theta(n)$ (a=3, b=4, c=2)
    - $f(n) = 1000n^2+100n-6$ : $f(n) = \Theta(n^2)$
    - $f(n) = 6 \cdot 2^n + n^2$ : $f(n) = \Theta(2^n)$

# 순환

- 순환 (recursion)
  - 정의하려는 그 개념 자체를 정의 속에 포함
  - 종류
    - 직접 순환 : 함수가 직접 자신을 호출
    - 간접 순환 : 다른 제 3의 함수를 호출하고 그 함수가 다시 자신을 호출
  - 순환 방식의 적용
    - 분할 정복(divide and conquer)의 특성을 가진 문제에 적합
    - 어떤 복잡한 문제를 직접 간단하게 풀 수 있는 작은 문제로 분할하여 해결하려는 방법.
    - 이 작고 간단한 문제는 원래의 문제와 그 성질이 같기 때문에 푸는 방법도 동일
  - 순환 함수의 명령문 골격
    - if simplest case then solve directly;<br>else make a recursive call to a simpler case;

# 순환 함수의 예

- 이진 탐색
  - 정의 : 주어진 탐색키 key가 저장된 위치(인덱스) 찾아내는 방법
    - key = a[mid] : 탐색 성공, return mid
    - key < a[mid] : a[mid]의 왼편에 대해 탐색 시작
    - key > a[mid] : a[mid]의 오른편에 대해 탐색 시작
  - 순환 함수로 표현

```
binarySearch(a[], key, left, right)
   // a[mid] = key인 인덱스 mid를 반환
   if (left <= right) then {
      mid ← (left + right) / 2;
      case {
         key = a[mid] : return mid;
         key < a[mid] : return binarySearch(a, key, left, mid-1);
         key > a[mid] : return binarySearch(a, key, mid+1, right);
      }
   }
   else return -1;      // key 값이 존재하지 않음
end binarySearch()
```

# 순환 알고리즘과 점화식(1)

- $C_N = C_{N-1} + N, N \geq 2, C_1 = 1$

$$C_N = C_{N-1} + N$$
$$= C_{N-2} + (N-1) + N$$
$$= C_{N-3} + (N-2) + (N-1) + N$$
$$\ldots$$
$$= C_1 + 2 + \ldots + (N-2) + (N-1) + N$$
$$= 1 + 2 + \ldots + (N-2) + (N-1) + N$$
$$= \frac{N(N+1)}{2}$$

➡ $C_N = O(N^2)$

# 순환 알고리즘과 점화식(2)

- $C_N = C_{N/2} + 1, N \geq 2, C_1 = 0$

  $N = 2^n$으로 가정하면,

  $C_{2^n} = C_{2^{n-1}} + 1$

  $= C_{2^{n-2}} + 1 + 1$

  $= C_{2^{n-3}} + 3$

  …

  $= C_{2^0} + n$

  $= n$

➡ $C_N = O(logN)$

# 순환 알고리즘과 점화식(3)

- $C_N = C_{N/2} + N \ , N \geq 2, C_1 = 0$

  $N = 2^n$으로 가정하면,

  $C_{2^n} = C_{2^{n-1}} + 2^n$

  $= C_{2^{n-2}} + 2^{n-1} + 2^n$

  $= C_{2^{n-3}} + 2^{n-2} + 2^{n-1} + 2^n$

  $\ldots$

  $= C_{2^0} + \ldots + 2^{n-2} + 2^{n-1} + 2^n$

  $\approx 2N$

※ 무한등비급수의 합 공식

$$S = \frac{a}{1-r}$$

초항 $a = N$이고 공비 $r = \frac{1}{2}$인 무한등비급수

$$C_N = N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \ldots$$

$$S = \frac{N}{1-\frac{1}{2}} = \frac{N}{\frac{1}{2}} = 2N$$

➡ $C_N = O(N)$

# 순환 알고리즘과 점화식(4)

- $C_N = 2C_{N/2} + N \ , N \geq 2, \ C_1 = 0$

  $N = 2^n$으로 가정하면,

  $C_{2^n} = 2C_{2^{n-1}} + 2^n$

  $\dfrac{C_{2^n}}{2^n} = \dfrac{C_{2^{n-1}}}{2^{n-1}} + 1$

  $= \dfrac{C_{2^{n-2}}}{2^{n-2}} + 1 + 1$

  ...

  $= n$

  $C_{2^n} = 2^n \cdot n$

➡ $C_N = O(NlogN)$

# 순환 알고리즘과 점화식(5)

- $C_N = 2C_{N/2} + 1$ , $N \geq 2$, $C_1 = 1$

  $N = 2^n$으로 가정하면 ,

  $$C_{2^n} = 2C_{2^{n-1}} + 1$$

  $$\frac{C_{2^n}}{2^n} = \frac{C_{2^{n-1}}}{2^{n-1}} + \frac{1}{2^n}$$

  $$= \frac{C_{2^{n-2}}}{2^{n-2}} + \frac{1}{2^{n-1}} + \frac{1}{2^n}$$

  …

  $$= 1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n}$$

  $$C_{2^n} = 2^n + \frac{2^n}{2} + \frac{2^n}{4} + \ldots + 1$$

  $$C_N \approx N + \frac{N}{2} + \frac{N}{4} + \ldots = 2N$$

$$C_N = O(N)$$