

# 프로그래밍 언어론

1.1 Scheme Programming

컴퓨터공학과  
이만호



# Scheme Programming

## 학습 목표

Recursion을 이용하여 프로그램을 작성하는 개념에 대해서 학습하고, 다양한 예제 프로그램을 통해 실제 적용 사례에 대해 학습한다.

## 학습 내용

- Recursion 없는 Program
- Recursion의 특성
- Recursion on Numbers
- Recursion on Lists



# 목 차

- 들어가기
- 학습하기
  - Recursion 없는 Program
  - Recursion의 특성
  - Recursion on Numbers
  - Recursion on Lists
- 평가하기
- 정리하기



# 알고가기



아래에 열거한 것 중에서 recursion 개념이 적용된 작업으로 보기 어려운 것은?

- a. 피자 한 판을 가지고 16사람이 나누어 먹으려고 중심을 지나는 선을 따라 8번 잘라서 16조각을 얻었다.
- b. 종이를 2등분 하고, 2등분된 조각을 포개서 다시 2등분 하는 작업을 반복하면, 여러 장의 같은 크기의 종이 조각을 얻을 수 있다.
- c. 어느 대학교에서 강의실의 개수를 파악하려고, 단과대학에 지시하고, 단과대학에서는 학과에 지시하여 파악하도록 하였다.
- d. 오름차순으로 정렬되어 있는 배열에서 원하는 값을 찾기 위해 binary search 알고리즘을 적용했다.

확인



# | Recursion 없는 Program

## Number를 다루는 Program

`(max3 n1 n2 n3)` : 3개의 숫자를 parameter로 받아 최대값을 return한다.

## List를 다루는 Program

`(third ls)` : 3개 이상의 item을 가진 list를 parameter로 받아 3번째 item을 return한다.

`(reverse3 ls)` : 3개의 item을 가진 list를 parameter로 받아 item의 순서가 거꾸로 된 list를 return한다.

## 기타 Program

`(type-of item)` : item 하나를 parameter로 받아 item의 type이 무엇인지 판정한 결과를 return한다.



# | (max3 n1 n2 n3)

3개의 숫자를 parameter로 받아 최대값을 return한다.

```
(DEFINE (max3 n1 n2 n3)
  (if (> n1 n2)
    (if (> n1 n3)
      n1
      n3)
    (if (> n2 n3)
      n2
      n3)))

(DEFINE (max3 n1 n2 n3)
  (COND
    ((> n1 n2) (cond ((> n1 n3) n1)
                     (ELSE n3)))
    ((> n2 n3) n2)
    (ELSE      n3)))
```



## | (third ls)

**ls:** 3개 이상의 item을 가진 list

**ls의 item 중에서 3번째 item을 return한다.**

```
(third '(a b c d e)) → c
```

```
(DEFINE (third ls)  
  (CAR (CDR (CDR ls))))
```

```
(DEFINE (third ls)  
  (CADDR ls))
```



## | (reverse3 ls)

ls: 3개의 item을 가진 list

ls의 item의 순서가 거꾸로 된 list를 return한다.

```
(reverse3 '(a b c)) → (c b a)
```

```
(DEFINE (reverse3 ls)
  (CONS (CADDR ls)          ; (3rd ls)
    (CONS (CADR ls)         ; (2nd ls)
      (CONS (CAR ls) '())))) ; (1st ls)
```





## | (type-of item)

item: 임의의 item

item의 type이 무엇인지 판정한 결과를 return한다.

```
(type-of 34) → number  
(type-of 'a) → symbol  
(type-of '(a b c)) → pair  
(type-of '()) → null-list  
(type-of (+ 3 4)) → number  
(type-of (car '((a)))) → pair
```

```
(DEFINE (type-of item)  
  (COND  
    ((NUMBER? item) 'number)  
    ((SYMBOL? item) 'symbol)  
    ((PAIR? item) 'pair)  
    ((NULL? item) 'null-list)  
    (ELSE 'some-other-type)))
```




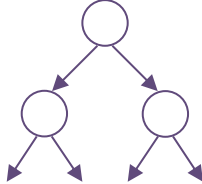
# | Recursion

## Recursive Definition

➔ 문제의 정의가 원래 자신의 문제를 다시 기술하고 있다.

예

### Recursion

- $n! = n*(n-1)!$     where  $0! = 1$
- $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$     where  $\text{fib}(0) = 0, \text{fib}(1) = 1$
- $C(n,r) = C(n-1,r) + C(n-1,r-1)$     where  $C(n,n) = C(n,0) = 1$
- List 
- Tree 
- 점화식

$$a_n = 2*a_{n-1} + 1 \quad \text{단} \quad a_0 = 0$$

## Recursive Function

➔ Function 실행 과정에서 자기 자신을 호출한다.



# | Recursion으로 문제 해결하기

## Recursion으로 해결할 수 있는 문제의 특성

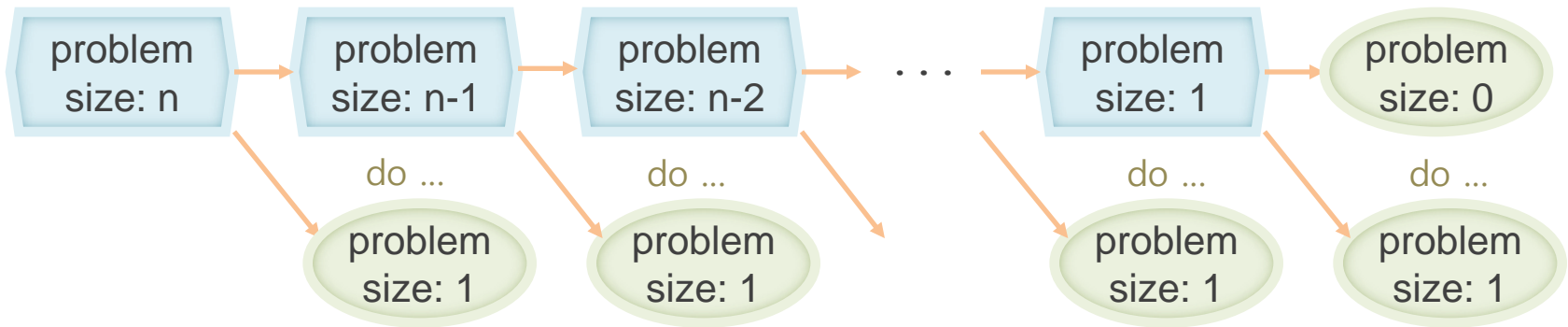
- ➔ Base case(simple case, trivial case)가 있다.
  - ▶ 문제가 매우 간단하여 답을 직관적으로 알 수 있는 경우를 말함.
- ➔ Base case가 아닌 경우 (문제의 크기가 크다)
  - ▶ 크기가 큰 문제를 작은 단위로 나누어 처리할 수 있다.
  - ▶ 작은 단위로 나누어 처리한 결과를 통합하여 원래 문제의 결과를 얻을 수 있다.
  - ▶ 크기가 큰 문제나 작은 문제나 해결 방법은 동일하다.
  - ▶ 작은 단위로 나누는 과정을 반복하면 결국 base case에 도달한다.

```
any-type fun(Prob)
{
    if (Prob is the base case)
        return (직관적으로 알고 있는 답);
    else
        return Combine(fun(Prob-1), fun(Prob-2));
}
```



# | Recursion의 구체적 개념

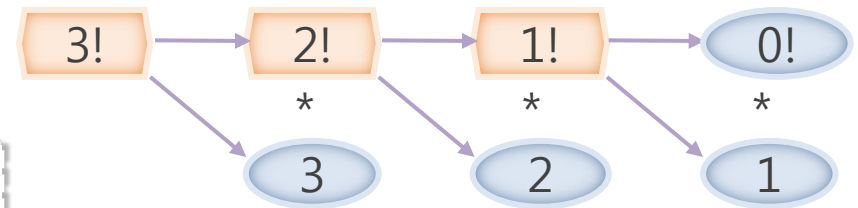
## 전형적인 Recursion 접근 방법



## n!의 경우

$$n! = n \cdot (n-1)! \quad \text{where } 0! = 1$$

```
(DEFINE (fact n)
  (IF (= n 0)
    1
    (* n (fact (- n 1)))))
```



# | Recursion으로 문제 해결하기

## 전형적인 recursion 접근 방법

➔ Parameter로 받은 문제의 크기가  $n$ 일 때

- ▶ Base case:  $n=0$
- ▶ Recursion: 문제의 크기가  $(n-1)$ 인 경우

**(sum n)** :  $1 + 2 + \dots (n-1) + n$

```
sum(n) = {1 + 2 + ... (n-1)} + n
sum(n) = n + sum(n-1)      where sum(0)=?
```

```
(DEFINE (sum n)
  (IF (= n 0)
    0
    (+ n (sum (- n 1))))))
```

**$n!$**  =  $1 * 2 * \dots * (n-1) * n$

```
fact(n) = {1 * 2 * ... * (n-1)} * n
fact(n) = n * fact(n-1)    where fact(0)=?
```



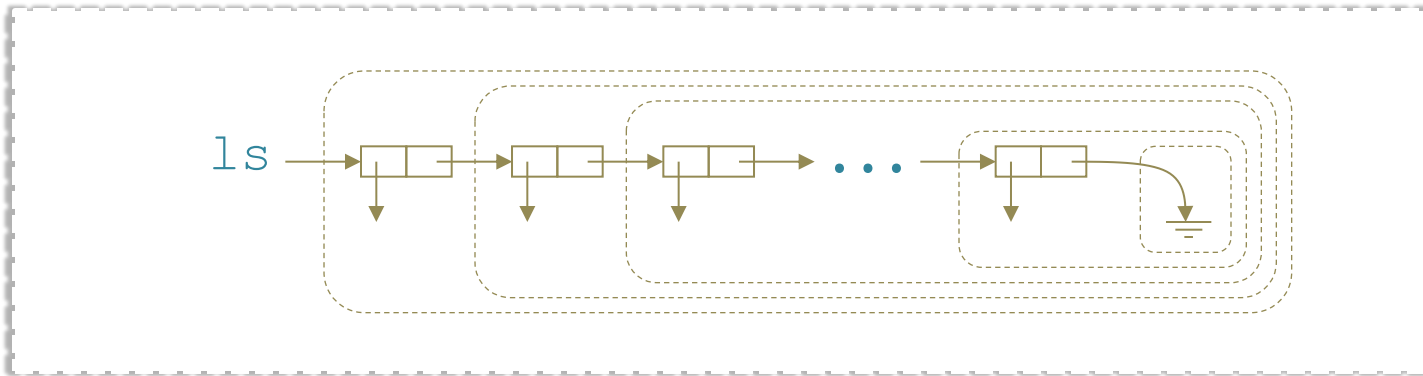
# | Recursion on Lists

## 전형적인 recursion 접근 방법

### ➔ Parameter로 `ls`를 받았을 때

- ▶ Base case: `ls`가 null list
- ▶ `(CAR ls)`를 적절히 처리
- ▶ Recursion on `(CDR ls)`

※ `list`에 대한 정보는 맨 처음 `node`의 `pointer`만 알고 있음.



# | (member? item ls)

ls: 중첩되지 않은 list

item이 ls의 member이면 #T를 return한다.

```
(member? 7 '(1 3 5 7 9)) → #T
(member? 'a '(my cat has kittens)) → #F
(member? 'foo '()) → #F
```

- Base case: ls가 null → #F
- When ls is not null and (CAR ls)==item → #T
- When ls is not null and (CAR ls)!=item
  - ▶ item과 같은 것을 아직 못 찾았으니 (CDR ls)에서 더 찾아야 함
 

```
(member? item (CDR ls))
```

```
(DEFINE (member? item ls)
  (COND
    ((NULL? ls) #F)
    ((EQ? item (CAR ls)) #T)
    (ELSE (member? item (CDR ls)))))
```



# | (equalList? ls1 ls2)

ls1, ls2: 중첩되지 않은 list

ls1과 ls2가 내용이 동일하면 #T를, 아니면 #F를 return한다.

```
(equalList? '(a b) '(a b)) → #T
(equalList? '(a b) '(a b c)) → #F
(equalList? '(a b) '(a d)) → #F
```

→ Base case: ls1과 ls2가 모두 null → #T

→ When ls1과 ls2가 모두 null 이 아니고,

▶ (CAR ls1) != (CAR ls2)이면 → #F를 return

(CAR ls1) == (CAR ls2)이면 → (CDR ls1)과 (CDR ls2)를  
더 비교해야 함. → (equalList? (CDR ls1) (CDR ls2))

```
(DEFINE (equalList? ls1 ls2)
  (COND
    ((NULL? lis1) (NULL? lis2))
    ((NULL? lis2) #F)
    ((EQ? (CAR ls1) (CAR ls2))
      (equalList? (CDR ls1) (CDR ls2)))
    (ELSE #F)))
```





# | (append ls1 ls2)

ls1, ls2: 중첩되지 않은 list

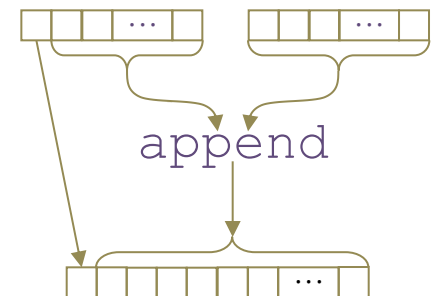
ls1과 ls2의 모든 item들을 포함하는 list를 return한다.

```
(append '(a b c) '(d e)) → (a b c d e)
(append '() '(d e)) → (d e)
(append '(a b c) '()) → (a b c)
```

→ Base case: ls1 = null → ls2

→ When ls is not null,

```
(CONS (CAR ls1)
      (append (CDR ls1) ls2))
```



```
(DEFINE (append ls1 ls2)
  (COND
    ((NULL? ls1) ls2)
    (ELSE (CONS (CAR ls1)
                  (append (CDR ls1) ls2)))))
```



# | (length ls)

ls: 중첩되지 않은 list

ls의 모든 item의 개수를 return한다.

```
(length '()) → 0
(length '(a b c)) → 3
```

→ Base case: ls가 null → 0

→ When ls is not null,

(length (CDR ls))의 결과 + 1

```
(DEFINE (length ls)
  (IF (NULL? ls)
      0
      (+ 1 (length (CDR ls)))))
```



## | (sum ls)

ls: number로만 구성된 중첩되지 않은 list

ls의 모든 item의 값을 더한 결과를 return한다.

```
(sum '(3 8 6)) → 17  
(sum '()) → 0
```

→ Base case: ls가 null → 0

→ When ls is not null,

(sum (CDR ls))의 결과 + (CAR ls)

```
(DEFINE (sum ls)  
  (IF (NULL? ls)  
      0  
      (+ (CAR ls) (sum (CDR ls))))))
```



# | (nth n ls)

**n**: 양의 정수

**ls**: 중첩되지 않은 list,  $(\text{length } ls) > n$

→ 첫째 item이 0번째 item이라고 생각한다.

**ls**에서 **n**번째 item을 return한다.

```
(nth 2 '(a b c d)) → c
(nth 0 '(a b c d)) → a
```

→ Base case:  $n=0 \rightarrow (\text{CAR } ls)$

→ When  $n > 0$ ,  
 (CDR ls) 에서  $n-1$ 번째 item을 return한다.

```
(DEFINE (nth n ls)
  (IF (= n 0)
      (CAR ls)
      (nth (- n 1) (CDR ls))))
```



# | (even1 ls)

**ls**: number로 구성된 중첩되지 않은 list.

**ls**에서 짝수만으로 구성된 list를 return한다.

```
(even1 '(3 8 6 5 9 4)) → (8 6 4)
(even1 '()) → ()
```

→ Base case: ls가 null → ()

→ When ls is not null,

(CAR ls)가 even이면, → (CONS (CAR ls) (even1 (CDR ls)))

(CAR ls)가 even이 아니면, → (even1 (CDR ls))

```
(DEFINE (even1 ls)
  (COND ((NULL? ls) '())
        ((EVEN? (CAR ls))
         (CONS (CAR ls) (even1 (CDR ls))))
        (ELSE (even1 (CDR ls)))))
```



# | 중첩된 List에 대한 Recursion

## 중첩되지 않은 list(*ls*라고 하자)에 대한 recursion

→ (CDR *ls*)에 대해서만 recursion을 적용하면 됨

## 중첩된 list(*ls*라고 하자)에 대한 recursion

→ (CDR *ls*)에 대해서 recursion을 적용함

→ (CAR *ls*)가 atom인 경우, 중첩되지 않은 경우와 동일하게 처리함

→ (CAR *ls*)가 list인 경우, (CAR *ls*)에 대해서도 recursion을 적용해야 함

```
(DEFINE (sum* ls)
  (COND
    ((NULL? ls) 0)
    ((NOT (PAIR? (CAR ls)))
      (+ (CAR ls) (sum* (CDR ls))))
    (ELSE (+ (sum* (CAR ls)) (sum* (CDR ls))))))
```



# 평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?  
총 2문제가 있습니다.

START



## 평가하기 1

1. 아래에 주어진 Scheme 프로그램은 number로 구성된 list를 parameter로 받아 제일 처음 나오는 양수를 return하는 프로그램이다. 밑줄 위에 들어갈 수 없는 것은? 양수가 없으면 null을 return한다.

```
(DEFINE (pos ls)
  (COND ((NULL? ls) '())
        (_____ (CAR ls))
        (ELSE (pos (CDR ls)))))
```

- a. (NOT (NEGATIVE? (CAR ls)))
- b. (POSITIVE? (CAR ls))
- c. (> (CAR ls) 0)
- d. (NOT (<= (CAR ls) 0))

확인





## 평가하기 2

2. 아래에 주어진 Scheme 프로그램은 임의의 list를 parameter로 받아, number로만 구성된 list를 생성해서 return하는 프로그램이다. 밑줄 위에 들어갈 수 있는 것은?

```
DEFINE (numL ls)
  (COND ((NULL? ls) '())
        ((number? (CAR ls)) _____)
        (ELSE (numL (CDR ls)))))
```

- a. (CONS (CDR ls) (numL (CAR ls)))
- b. (CONS (CAR ls) (numL (CDR ls)))
- c. (numL (CDR ls))
- d. (numL (CAR ls))

확인



# 정리하기

## ➡ Recursion 없는 Program

## ➡ Recursion의 특성

- ▶ Base case(simple case, trivial case)
- ▶ 크기가 큰 문제를 작은 단위로 나누어 처리하고 결과를 통합함

## ➡ Recursion on Numbers

- ▶ Base case:  $n=0$
- ▶ Recursion: 문제의 크기가  $(n-1)$ 인 경우

## ➡ Recursion on Lists

- ▶ Base case: null list
- ▶ (CAR  $1s$ )를 적절히 처리
- ▶ Recursion on (CDR  $1s$ )



“ 강의를 마치겠습니다. 수고하셨습니다. ”

