프로그래밍 언어론

Scheme 기초
(Scheme Basics)

컴퓨터공학과
이만호

충남대학교
CHUNGNAM NATIONAL UNIVERSITY

Scheme 기초

학습목표

Scheme으로 프로그래밍 하는데 필요한 Scheme에 대한 기초 지식을 학습한다.

학습내용

- Scheme의 특성
- Scheme의 언어 체계
- Data Types
- Primitive List functions
- IF/COND Expression

목 차

- 들어가기
- 학습하기
  - Scheme의 특성
  - Scheme의 언어 체계
  - Data Types
  - Primitive List functions
  - `IF/COND` Expression
- 평가하기
- 정리하기

알고가기

Scheme의 recursive function에서 주로 처리하는 대상이 되는 자료구조의 형태는?

a. Integer
b. Real
c. Symbol
d. List

확인

Scheme의 특성

- Lisp으로부터 유래
- 교육 목적으로 많이 사용됨
  - 매우 간단함 (syntax, semantics)
  - 무료로 제공되는 interpreter가 많음
- Static scope rule만 적용 (dynamic scope rule은 지원 안함)
- 변수의 type 선언이 없음 (typeless)
- Function이 first-class object(또는 entity) 임
  - Expression/Function 의 결과가 될 수 있음
  - List의 element가 될 수 있음
  - Parameter로 전달 가능
  - 변수에 assign될 수 있음 (바람직하지 않음)
- Prefix 표현 사용
  - `(+ a b)`, `(combination n r)`

Scheme Interpreter

Scheme Interpreter의 실행 방법

- Read-Evaluate-Write 를 무한히 반복한다.

1. **Prompt를 화면에 출력** (prompt를 '>'라고 하자)
2. **Read an expression**
   - 사용자가 입력하는 expression(수식)을 list 형태로 읽는다.
   - 입력되는 expression
     - Function definition(정의)
     - Function application(적용, 호출)
3. **Evaluate (function application의 경우)** : function EVAL에 의해서 실행됨
   1. 전달 받은 parameter 값을 계산한다. (계산 순서는 정해져 있지 않음)
   2. 계산된 parameter 값이 body 내의 parameter를 대체한다.
   3. Body를 실행한다.
   4. Body에서 마지막으로 계산한 expression의 값을 결과로 return한다.
4. **Write**
   - 결과 값을 화면에 출력한다.

Scheme의 언어 체계

Alphabet

- 출력이 가능한 모든 문자

Word(Lexicon)

- **Number**
  - integer, real
- **Identifier**
  - function 이름, 변수 이름, …
- **Symbol**
  - Quoted identifier

Expression

- **Number**
- **Identifier**
- **Symbol**
- **Application**
  - (operator operand-1 operand-2 …)

명령형언어의 체계

- Alphabet
  - –영문자, 숫자, 특수문자
- Lexicon
  - –Number, Identifier, symbol
- Expression
  - –산술식, 관계식, 논리식
- Statement
  - –할당문, …
- Subprogram
  - –Procedure, Function

Numbers

Integers

- 부호가 사용될 수 있음

```
375, +375, -375
```

- 자릿수에 제한이 없음

```
732479274298366425396436700747752869889247924…4
```

  - 2진수 표현을 사용하지 않음

Real

- 부호가 사용될 수 있음

```
  3.8,   +3.8,   -3.8,      247.,   +247.,    -247.
.247,  +.247,  -.247,    0.247,  +0.247,   -0.247
```

- 지수 표현(exponential notation)

```
  123.456e20,    1.23456e22,    0.00123456e-25
+123.456e20,   +1.23456e22,   +0.00123456e-25
-123.456e20,   -1.23456e22,   -0.00123456e-25
```

Identifiers

출력이 가능한 모든 문자들의 조합

사용될 수 없는 문자

- 특별한 의미를 가지는 문자: ( ) [ ] ' ` , @ " \ # ;
- 제어 문자: \t, \n, ...
- 공백(space character)

첫 문자로 사용될 수 없는 문자

0~9 . + -

예

```
foo   woman   blsp   what?   b2*p

top-down   principle+interest   *big*

i<j   i-j   i.j   =i   i*j   i&j   ...
```

Symbols

문자상수

- **Quoted identifier: identifier 앞에 quote 문자를 붙인다.**
  - quote function을 사용한 것과 동일하다.
- **Case-sensitive하지 않다.**
- **길이에 제한이 없다.**

예

```
'pi == (quote pi)
'WOMBAT == 'wombat == (quote WOMBAT) == (quote wombat)
'long_symbol-with_funny%characters+:~!
```

quote 문자를 제거하면 모두 identifier가 된다.

```
'748 == (quote 748) == 748  ; 'number: symbol의 의미가 없다.
```

Expressions

Number : integer, real

```
398739593    3987.39593
```

Symbol : quoted identifier : (quote E) or 'E

```
'foo    (quote foo)
```

- E를 계산하지 않고, E를 결과로 return한다.

Variable(변수) : 어떤 값을 저장하기 위한 identifier

```
foo
```

- First-class object이면 variable에 저장할 수 있다.

Application(함수의 적용)

```
(function argument_1 ...)
(+ 3 8)
(factorial 3)
(a b c)     ; a는 반드시 function이어야 함
```

Variable의 값 정의하기

Primitive function DEFINE을 사용한다.

Form : `(DEFINE V E)` ; expression

- **V** : identifier. Variable(변수)로 사용됨
- **E** : expression

➔ **E를 계산하고 그 결과를 V에 저장한다.**

▶ "V는 E의 결과로 bound된다"라고 표현한다.

➔ **(DEFINE V E)의 실행 결과는 지정되지 않음(unspecified)**

예

```
(DEFINE pi 3.1415928)    ; pi: 3.1415928
(DEFINE qi pi)           ; qi: 3.1415928
(DEFINE animal 'tiger)   ; animal: tiger
```

Data Types

Atom

- List가 아닌 모든 것. (즉, cons cell이 없다. Pair가 아니다.)
- number, symbol, string, boolean
  - String: "...", ""(null string)
  - Boolean: #T, #F

List : structured data

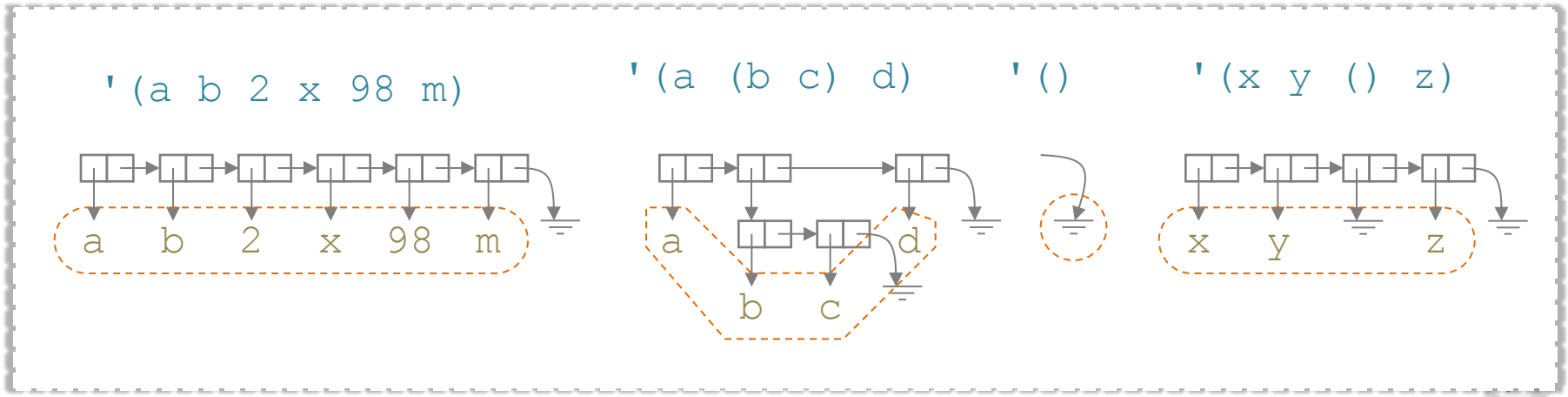
- Form : (data_1 ... data_n) **where n >= 0**
  - 모든 function application은 list이다.
  - Data로서의 list는 quote를 사용해야 한다.
- List 구조는 중첩될 수 있다.

예

```
(DEFINE vec '(5 8 3))     ; → unspecified,   vec:(5 8 3)
(DEFINE vec (quote (5 8 3)))
(+ 3 6)       ; → 9              function application
'(+ 3 6)      ; → (+ 3 6)        data로서의 list
'(Sat Sun) == (quote (Sat Sun))   ; → (Sat Sun)
'((Mon Tue Wed Thu Fri) (Sat Sun))
'()     ;     () : null list          ※ ()은 list이며 atom이기도 하다.
'(a (b c) () ((d e) ()) (f (g h) i) j)
```

List의 표현

- Linked list(연결리스트)로 표현한다.
- Linked list를 구성하는 각 node를 "cons cell" 또는 "pair" 라고 한다.
- Top-level item
  - 중첩되지 않은 최 상위에 있는 item
- List의 내부 구조



Primitive Functions

- Scheme이 기본적으로 제공하는 functions
- Numeric functions
  - Number를 다루는 function

```
+, -, *, /,
abs, sqrt, remainder, min, max, ...
```

  - 사용 방법

```
(+ 5 7)   (+ 5 7 8 6)   (+ 5)
(+)    ; → + 연산의 항등원
(*)    ; → * 연산의 항등원
```

- List functions
  - List를 다루는 function
- 기타
  - DEFINE, ...

Lambda Expression

무명 Function을 정의하는데 사용됨

Form : `(LAMBDA (parameters …) body)`

- `parameters(형식 매개변수)` : bound variables

실 매개변수를 전달하고 실행하기

```
((LAMBDA (parameters …) body) arguments …)
```

- `arguments:` 실 매개변수

예

```
(LAMBDA (x) (* x x))              ; → function
((LAMBDA (x) (* x x)) 7)          ; → 49
(LAMBDA (a b) (+ a b))
((LAMBDA (a b) (+ a b)) 24 5)     ; → 29
```

Function 이름

Lambda expression으로 무명 function을 정의

- Function 이름이 없으면 사용하기 불편함
- DEFINE을 이용하여 function에 이름을 줄 수 있다.

Form-1

```
(DEFINE function_name (LAMBDA (p …) body))
```

- 두번째 parameter인 lambda expression을 evaluate하고, 그 결과를 변수 function_name에 bind해 준다.

Form-2 : 교재에서 사용되고 있음

```
(DEFINE (function_name p …) body)
```

- Form-1과 의미가 동일함

예

```
(DEFINE (square x) (* x x))
    • square: (LAMBDA (x) (* x x))
(square 7)                    ; → 49
((LAMBDA (x) (* x x)) 7)      ; → 49
```

Primitive List Functions

Constructor(생성자)

- **List를 생성하는데 사용**
  - **CONS** : list를 생성함

Selector(선택자)

- **List에서 head와 tail을 선택하는데 사용**
  - **CAR** : list의 head를 선택함
  - **CDR** : list의 tail을 선택함 (could-er라고 읽음)

용어의 어원

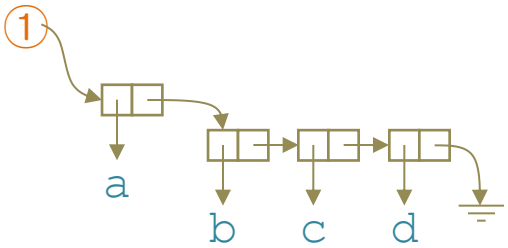
- CONS : CONStructing list
- CAR : Contents of the Address part of Register number
- CDR : Contents of the Decrement part of Register number
  - CAR과 CDR는 Lisp이 1950년대 후반에 최초로 설치된 IBM 704의 instruction임

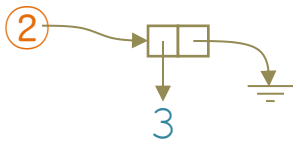
Constructor(생성자)

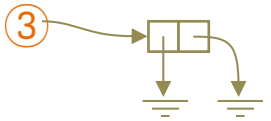
```
(CONS item list)
```

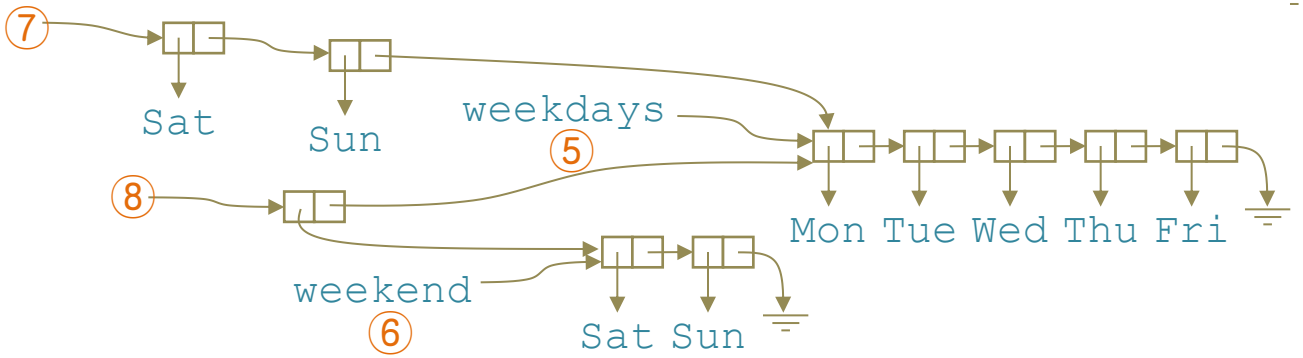
1. 하나의 cons cell을 할당한다.
2. Cons cell의 앞이 item을 point하도록 한다.
3. Cons cell의 뒤가 list를 point하도록 한다.

```
① (CONS 'a '(b c d))          ; → (a b c d)
② (CONS 3 '())                ; → (3)
③ (CONS '() '())              ; → (())
④ (CONS 1 (CONS 3 '()))       ; → (1 3)
⑤ (DEFINE weekdays '(Mon Tue Wed Thu Fri))
⑥ (DEFINE weekend '(Sat Sun))
⑦ (CONS 'Sat (CONS 'Sun weekdays))
⑧ (CONS weekend weekdays)
```









Selectors(선택자)

(CAR list)

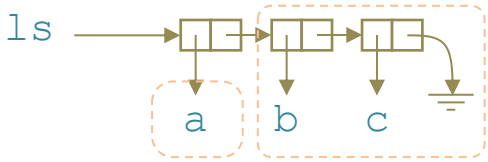
- `list`: non-empty list.
- `list`가 point하는 cons cell에서 앞의 pointer를 return한다.

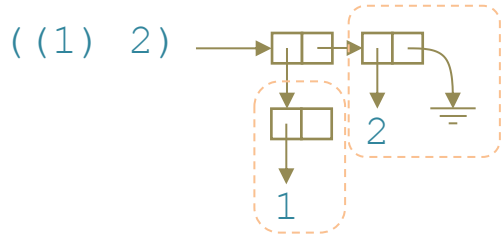
(CDR list)

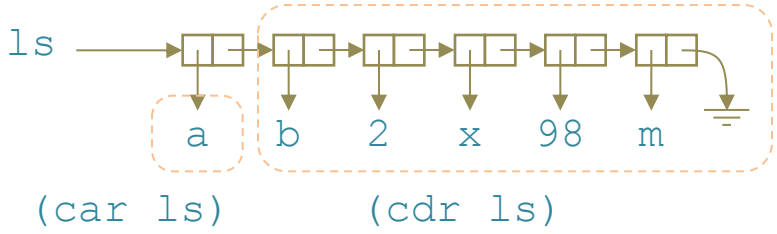
- `list`: non-empty list.
- `list`가 point하는 cons cell에서 뒤의 pointer를 return한다.
  - 결과는 항상 `list`이다.

예

```
(define ls '(a b c))
(car ls)   ==> a
(cdr ls)   ==> (b c)
(car '((1) 2))   ==> (1)
(cdr '((1) 2))   ==> (2)
```







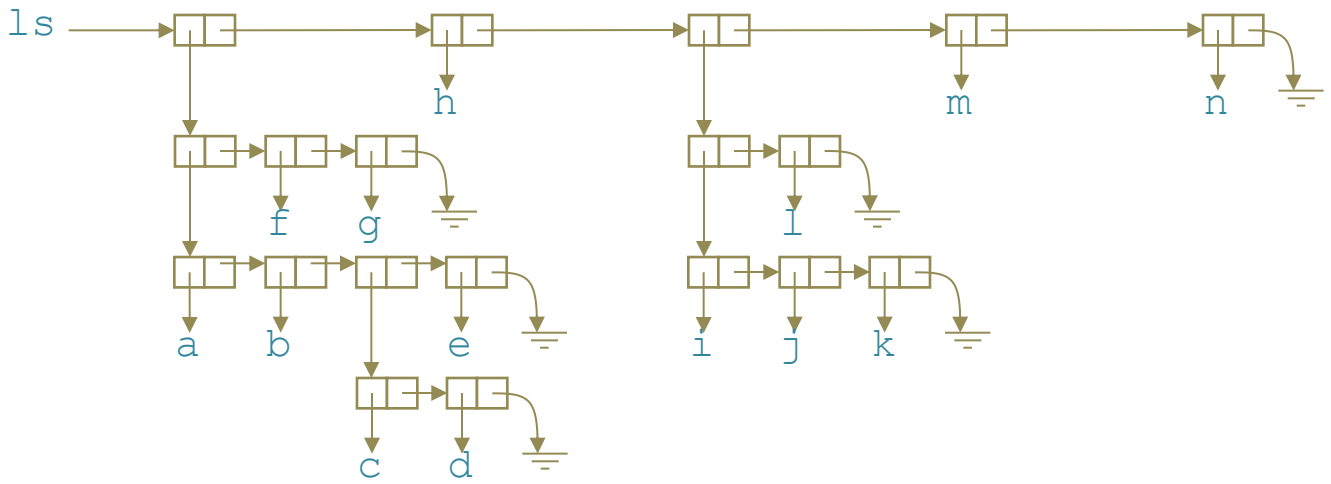
CAR와 CDR의 연속

CAR와 CDR는 연속해서 사용하는 경우가 많음

- 간단히 표현할 수 있는 방법을 사용
- (CAR (CAR (CDR (CDR E)))) ≡ (CAADDR E)
- (CDR (CAR (CAR E))) ≡ (CDAAR E)

예

```
(DEFINE ls '(((a b (c d) e) f g) h ((i j k) l) m n))
(CAAAR ls)
(CADDAAR ls)
```



CONS, CAR, CDR의 관계

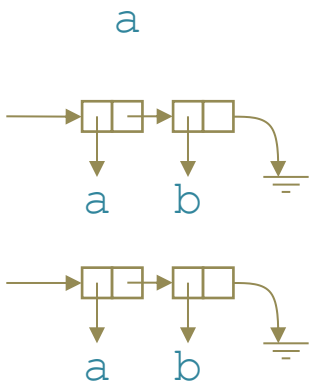
Constructor와 Selector는 밀접한 관계임

- `(CAR (CONS item list))` → `item`
- `(CDR (CONS item list))` → `list`
- `(CONS (CAR list) (CDR list))` → `list`

Predicates(조건 함수)

항상 boolean 값을 return하는 function

- Boolean 값: `#T`, `#F`



Type 관련 Predicates

```
(number? E)   (symbol? E)   (boolean? E)
(list? E)   (pair? E)   (null? E)
```

Number 관련 Predicates

```
(= n1 n2) (> n1 n2) (>= n1 n2)
(<> n1 n2) (< n1 n2) (<= n1 n2)
(ZERO? n) (POSITIVE? n) (NEGATIVE? n) (EVEN? n) (ODD? n)
```

Equivalence Predicates (동일성 test)

```
(= n1 n2)                : number의 동일성 test
(EQ? atom1 atom2)        :   atom의 동일성 test
(EQUAL? list1 list2)     :    list의 동일성 test
    (EQ?    'a 'a)           ; → #T
    (EQ?    '(a b) '(a b)) ; → #F
    (EQUAL? '(a b) '(a b)) ; → #T
```

| 논리 연산 함수

(OR E-1 ... E-n) where n≥0

- E-1부터 순서대로 값을 계산해서 #T인 E-k가 있으면 즉시 #T를 return한다.
- 모든 E-k가 #F이면 #F를 return한다.
- (OR) ; → or의 항등원

(AND E-1 ... E-n) where n≥0

- E-1부터 순서대로 값을 계산해서 #F인 E-k가 있으면 즉시 #F를 return한다.
- 모든 E-k가 #T이면 #T를 return한다.
- (AND) ; → and의 항등원

(NOT E)

IF Expression

IF expression

```
(IF Test
    Consequence      ; then part
    Alternative)     ; else part, optional
```

- Test의 결과가 #T이면, Consequence를 계산하고,
  #F이면, Alternative를 계산한다.
- 중첩될 수 있다.(nested)

예

```
(DEFINE (max2 a b) (IF (> a b) a b))
(max2 8 3) ; → (IF (> 8 3) 8 3) → 8
(max2 3 8) ; → (IF (> 3 8) 3 8) → 8
```

COND Expression

COND expression

```
(COND
    (Test_1   Consequence_1)
    (Test_2   Consequence_2)
    ...
    (Test_n   Consequence_n)
    (ELSE     Alternative))   ; optional
```

- 여러 개의 Test_k의 결과가 #T일 수 있다.
- Test_1부터 순서대로 값을 계산해서 #T인 Test_k가 있으면 Consequence_k 를 계산해서 그 결과를 return한다.

예

```
(DEFINE (max2 a b) (COND ((> a b) a)
                         (ELSE    b)))
    (max2 8 3) ; → (COND ((> 8 3) 8) (ELSE 3)) → 8
    (max2 3 8) ; → (COND ((> 3 8) 3) (ELSE 8)) → 8
```

COND Expression

COND expression

```
(COND
    (Test_1   Consequence_1)
    (Test_2   Consequence_2)
    ...
    (Test_n   Consequence_n)
    (ELSE     Alternative))   ; optional
```

- 여러 개의 Test_k의 결과가 #T일 수 있다.
- Test_1부터 순서대로 값을 계산해서 #T인 Test_k가 있으면 Consequence_k 를 계산해서 그 결과를 return한다.

예

```
(DEFINE (max2 a b) (COND    (> a b) a)
                            (< a b) b)
                            ( else  a)))
```

평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?

총 2문제가 있습니다.

START

평가하기 1

1. '(QUOTE)에 대한 설명으로 맞지 않는 것은?

a. 'foo는 atom이다.

b. 'foo는 (QUOTE foo)와 의미가 같다.

c. '(+ 3 4)의 결과는 '7이다.

d. '(foo goo hoo)에서, foo가 function일 수도 있다.

확인

평가하기 2

2. Scheme에서 atom으로 생각할 수 없는 것은?

a. 'foo

b. 314

c. '()

d. goo

확인

정리하기

Scheme의 특성

- Static scope rule만 적용
- 변수의 type 선언이 없음

Scheme의 언어 체계

- Alphabet
- Word(Lexicon)
- Expression

Data Types

- Atom, List

Primitive List functions

- Static scope rule만 적용
- 변수의 type 선언이 없음

IF/COND Expression

- Predicates 사용

"강의를 마치겠습니다. 수고하셨습니다."