프로그래밍의 원리 2012 가을
실습 1
cond, 재귀 함수, 함수를 인자로 받는 함수, 타입

서울대학교 프로그래밍 연구실
강동옥, 최민아

2012년 9월 13일

1. cond

   cond를 사용하면 조건에 따라 값을 선택할 수 있습니다. (cond (조건식 값)$^+$)로 쓸 수 있습니다.

```
(define a 3)
(define b (+ a 1))

(cond ((= a 4) 6)
      ((= b 4) (+ 6 7 a))
      (else 25))

(* (cond ((> a b) a)
         ((< a b) b)
         (else -1))
   (+ a 1))
```

   abs란 함수를 만들어 봅시다. 이 함수는 한 수를 입력으로 받아서 그 수의 절대값을 돌려 줍니다. 예를 들자면 다음과 같습니다.

```
(abs (- 4 6))
2
```

   abs 함수를 cond 명령어를 사용해 만들어 봅시다.

2. 재귀함수

   함수의 내부에서 자기 자신을 호출하는 함수를 '재귀 함수'라고 합니다. 재귀 함수가 항상 끝나려면 자기 자신을 호출할 때 원래의 인자보다 더 작은 인자로 불려야 하고 base case를 정의해야 합니다. 먼저 점화식으로 생각하는 것이 좋습니다.

   (a) sum

   sum(1) = 1
   sum(n) = n + sum(n-1)

   scheme 코드로 작성하면

```
(define (sum n)
    (if (= n 1)
        1
        (+ n (sum (- n 1)))
    ))
(sum 5)
(sum 0)
```

(b) fibonacci
fibo(1) = 1
fibo(2) = 1
fibo(n) = fibo(n-1) + fibo(n-2)

scheme 코드로 작성하면

```
(define (fibo n)
    (cond ((<= n 2) 1)
          (else ( ???? )))
    )
```

3. 함수를 인자로 받는 함수

   함수를 입력으로 받아서, 입력받은 함수를 호출하는 함수를 만들 수 있습니다. 다음 문장들을 실행기에 넣었을 때, 어떻게 될 지 생각해보고 실제로 입력하여 결과를 확인해 봅시다.

```
(define (sum term a next b)
    (if (> a b)
        0
        (+ (term a)
           (sum term (next a) next b))))

(define (inc n) (+ n 1))
(define (cube x) (* x x x))
(define (sum-cubes a b)
    (sum cube a inc b))

(define (identity x) x)
(define (sum-integers a b)
    (sum identity a inc b))
```

4. 이름 없는 함수

   lambda를 이용하여 이름 없는 함수를 만들 수 있습니다.

```
(lambda (r) (* 2 3.14 r))
```

   위의 함수는 반지름 r을 받아서 원의 둘레를 리턴합니다.

```
(define area-circle (lambda (r) (* 3.14 r r)))
(area-circle 5)
((lambda (f) (f 3)) area-circle)
```

연습문제

1. 팩토리얼을 계산해주는 함수 fact 를 만들어 봅시다. (fact n)은 n!을 계산합니다.

2. combination 함수를 만들어 봅시다. 이 함수는 정수 두 개 n, m을 입력으로 받아서 ${}_nC_m$를 계산해줍니다. 예를 들면 다음과 같습니다.

```
(combination 9 4)
126
```

   combination 을 두가지 방식으로 만들어 봅시다. 한번은 fact 함수를 사용해서 만들어 보고, 한번은 곱하기(×)와 나누기(/) 연산자를 사용하지 않고 만들어 봅시다. [1]

3. 한 개의 인자를 입력으로 받는 f, g 함수가 있을 때, f, g의 합성 함수란 $x \mapsto f(g(x))$를 나타냅니다. 두 개의 함수를 입력으로 받아 합성 함수를 만들어 주는 compose 함수를 만들어 봅시다.

```
(define (square x) (* x x))
(define (inc x) (+ x 1))
((compose square inc) 6)
49
```

4. 타입을 생각하며 프로그래밍 하는 연습을 해보겠습니다. 다음 코드에 대하여 수업시간에 배운 표기법을 이용하여 식의 타입을 주석으로 표기해 보세요.
   primitive type ι ::= int | real | bool | string | symbol | unit

```
(cons 4 'a) ; int X symbol
(display 2) ; unit
(lambda (a) (+ a 4)) ; int -> int
(cons 2 (cons "ab" 'd)) ; ..
(list 1 2 3) ; .. pair와 user-define 두 가지도
(define (foo x) (if x 1 5)) ; ..
(define (foo x y) (+ x y) ; ..
(define (bar x) (+ (car x) (cdr x))) ; ..
(define (bar x y) (if x #t (equal? y "abc"))) ; ..
(define (bar f x) (if (f x) (if (f (+ 3 x)) (- x 1) x) (* x 2))) ; ..
(define (sum a b) (if (= a b) b (+ a (sum (+ a 1) b))))
(define (memq item x)
  (cond ((null? x) false)
        ((eq? item (car x)) true)
        (else (memq item (cdr x)))
  )
) ; ..
```

5. 다음 식은 재귀적 함수 정의방법을 이용하여 어떤 숫자가 홀수인지 짝수인지 감별하고 있습니다. ...으로 표시된 빈칸을 채워 완성해 보세요.

[1] *힌트. 파스칼의 삼각형을 생각해보세요.*

```
(letrec (

          (is-even?
           (lambda (n)
             (or
              (zero? n)
              (is-odd? (sub1 n)
             )
            )
           )
          )

          (is-odd?
           (lambda (n)
             (and
              (...)
              (...)
             )
           )
          )

         )
         (is-odd? 14)
)
```

6. 2.(a) 예제의 sum 함수를 tail-recursive하게 바꿔 정의해 보세요.