# 프로그래밍의 원리 2012 가을 - 실습 4
# 계층별로 데이터 속구현 감추기, 여러 구현 동시에 지원하기

서울대학교 프로그래밍 연구실
강동옥

2012년 10월 11일

이번 실습의 목적은:

- 데이터의 속 구현을 프로그램 해본다
- 데이터의 속 구현을 여러가지 방식으로 프로그램 해본다
- 속 구현이 여럿인 경우에도 인터페이스만 알고 프로그램 하는 것을 익힌다

**연습문제**

1. 복소수(SICP 2.4.1)

   복소수 타입을 만들어 봅시다. 복소수는 두 가지 방식으로 나타낼 수 있습니다. 하나는 데카르트 좌표계 상의 직각 좌표 (x,y)로 나타낼 수 있고, 극좌표계를 써서 (r,θ)로 표현할 수 있습니다.

   위의 복소수 타입은 같은 모양의 데이터를 갖고 있습니다. 이 둘은 태깅을 써서 구분하는 것이 쉽습니다. 태깅하는 방법은 데이터와 함께 'rect 'polar 등과 같이 이름 을 써서 무슨 방식으로 만들어 주었는 지를 알려주는 방법입니다.

   복소수를 직각 좌표로 표현하는 복소수 타입을 만들어봅니다. c-rect-make는 실수부, 가수부 두 숫자를 받아서 직각 좌표 복소수를 타입을 만들어냅니다.

   $$\begin{aligned} \texttt{is-c-rect?} &: \alpha \to \mathit{bool} \\ \texttt{c-rect-make} &: \mathit{number} \times \mathit{number} \to \mathit{c\text{-}rect} \\ \texttt{c-rect-real} &: \mathit{c\text{-}rect} \to \mathit{number} \\ \texttt{c-rect-imaginary} &: \mathit{c\text{-}rect} \to \mathit{number} \end{aligned}$$

   복소수를 극좌표계로 표현하는 복소수 타입을 만들어봅니다. c-polar-make는 각도, 반경 두 숫자를 받아서 직각 좌표 복소수를 타입을 만들어냅니다.

| | | |
|---|---|---|
| `is-c-polar?` | : | $\alpha \rightarrow bool$ |
| `c-polar-make` | : | $number \times number \rightarrow c\text{-}polar$ |
| `c-polar-angle` | : | $c\text{-}polar \rightarrow number$ |
| `c-polar-radius` | : | $c\text{-}polar \rightarrow number$ |

이제 어떤 좌표계로 표현된 복소수라도 동작하게 하는 함수들을 만들어 봅시다. 이 함수들은 직각 좌표나 극 좌표계로 표현한 복소수 타입이 어떻게 구현되었는지에 상관없이 위에 제공된 함수들로만 만들어져야합니다.

| | | |
|---|---|---|
| `c-real` | : | $complex \rightarrow number$ |
| `c-imaginary` | : | $complex \rightarrow number$ |
| `c-angle` | : | $complex \rightarrow number$ |
| `c-radius` | : | $complex \rightarrow number$ |
| `c-conjugate` | : | $complex \rightarrow complex$ |

기본적으로 복소수의 계산 결과로 나온 값들은 직각 좌표계를 따른다고 합시다.

-참고

| | |
|---|---|
| $x = r\mathtt{c}os\theta$ | `cos` |
| $y = r\mathtt{s}in\theta$ | `sin` |
| $r = \sqrt{x^2 + y^2}$ | `sqrt,expt` |
| $\theta = \mathtt{a}rctan(y, x)$ | `atan` |

```
(define c1 (c-rect-make 1 2))
(define c2 (c-rect-make 3 4))
(define c3 (c-polar-make 0.7 3))
(define c4 (c-polar-make 0.5 2))
(c-rect-real c1)
1
(c-rect-imaginary c2)
4
(c-polar-angle c3)
0.7
(c-polar-radius c4)
2
(is-c-rect? c1)
#t
(is-c-rect? c3)
```

```
#f
(is-c-polar? c4)
#t
(c-real c1)
1
(c-real c3)
2.2945265618534654;(* 3 (cos 0.7))
(c-imaginary c2)
4
(c-imaginary c4)
0.958851077208406;(* 2 (sin 0.5))
(c-angle c1)
1.1071487177940904;(atan 2 1)
(c-angle c3)
0.7
(c-radius c2)
5;(sqrt (+ (expt 3 2) (expt 4 2)))
(c-radius c4)
2
(define c5 (c-conjugate c1))
(define c6 (c-conjugate c3))
(c-real c5)
1
(c-imaginary c5)
-2
(c-real c6)
2.2945265618534654
(c-imaginary c6)
-1.932653061713073
```

2. $n$명의 학생을 1번부터 $n$번까지 번호를 매긴 후 난수발생기를 통해 추출한 $m$개의 자연수 중 어떠한 수로도 나누어 떨어지지 않는 번호를 가진 학생에게만 A학점을 주기로 했습니다. 학생 수 $n$과 숫자리스트를 입력받아 A학점을 받을 학생들의 번호 리스트를 반환하는 함수 `lazy-ta`를 정의해 봅시다.

   이 문제를 직접 해결하려 하지 말고 독립적인 기능을 하는 여러 함수를 만들어서 문제를 푸는데 이용해 봅시다.

   (a) 우선 1부터 n까지의 정수를 원소로 갖는 리스트를 만드는 함수 `make-nlist`를 만들어 봅시다.

   ```
   (define (make-nlist n) ...)
   (make-nlist 8)
   (1 2 3 4 5 6 7 8)
   ```

   (b) 리스트를 인자로 받아서 2로 나누어 떨어지는 원소들만 리스트로 만들어서 돌려주는 `get` 함수를 만들어 봅시다.

   ```
   (define (get-2 lst) ...)
   (get-2 (make-nlist 8))
   (2 4 6 8)
   ```

   (c) `get-2` 함수를 일반화 시켜서 함수 $f$ 를 인자로 하나 더 받아 리스트의 원소를 $x$ 라 할 때, `(f x)`를 만족하는 원소들만 리스트로 만들어서 돌려주는 함수 `get`을 만들어 봅시다.

   ```
   (define (get lst f) ...)
   (get (make-nlist 8) even?)
   (2 4 6 8)
   ```

   (d) 정수 $n$과 정수 리스트 $lst$를 인자로 받아서 정수가 리스트의 어떤 정수로도 나누어 떨어지지 않으면 `#t`를 반환하는 함수 `notdiv?`를 만들어 봅시다.

   ```
   (define (notdiv? n lst) ...)
   (notdiv? 100 '(2 3 5))
   #f
   ```

   (e) 위에서 만든 함수들을 조합해서 `lazy-ta`를 만들어 봅시다. 주의할 점은 `get`함수의 인자로 주어지는 함수는 인자를 하나 받지만, `notdiv?`는 두개를 받습니다. 또한 `notdiv?`에 넘어갈 인자는 항상 `lazy-ta`에서 넘어오는 인자로 고정되기 때문에 이를 이용하여 인자를 하나 받는 함수를 만들 수 있습니다.

```
(lazy-ta 100 (list 2 3 5))
(1 7 11 13 17 19 23 29 31 37 41 43 47 49 53 59 61 67 71 73 77 79 83 89 91 97)
(lazy-ta 50 (list 3 4 5))
(1 2 7 11 13 14 17 19 22 23 26 29 31 34 37 38 41 43 46 47 49)
```