프로그래밍의 원리 2011 가을 - 실습 3
사용자 정의 타입, 여러 가지 방법으로 데이터 구현하기

서울대학교 프로그래밍 연구실
최민아

2012년 9월 27일

이번 실습의 목적은:

- 데이터의 속 구현을 프로그램 해본다
- 데이터의 속 구현을 여러가지 방식으로 프로그램 해본다

주의사항 모든 연습문제는 타입을 따지면서 프로그램 하는 습관을 들이도록 합시다.

연습문제

1. 사용자 정의 타입 - 고정 소수점

   소수점 아래 2자리까지를 표현할 수 있는 고정 소수점 타입을 만들어 봅시다. 고정 소수점은 실수를 정수와 소수점 이하 부분으로 나누어서 표현하는 방식입니다. 소수점 이하 부분을 무한히 표현할 수가 없기 때문에 크기를 제한합니다. 여기서는 소수점 부분이 0에서 99까지의 정수를 갖는다고 하겠습니다. 음수의 경우에는 소수점 부분이 항상 0과 99 사이에 있도록 정수 부분의 숫자를 조정해 입력이 들어옵니다. (예: -3.10의 경우 (fixed-make -4 90))

   고정 소수점을 표현하기 위해서는 두 개의 정수를 저장할 공간이 필요합니다. 두 개의 정수를 저장하는 방법은 여러가지가 있을 수 있습니다. 예를 들어, 실수 3.04는 순서쌍 (cons 3 4)으로 표현할 수도 있고, '(3 0 4)처럼 리스트로 나타낼 수도 있습니다. (list (cons 'r 3) (cons 'd 0 4))처럼 각각의 숫자에 표시를 해 둔 순서쌍의 리스트로 나타낼 수도 있습니다. 내부를 어떻게 구현하는지는 자유입니다.

   (a) 고정 소수점 값 만들기

   정수 부분, 소수점 부분 두 개의 수를 받아서 고정소수점 타입의 값을 돌려주는 함수 `fixed-make`를 만들어 봅시다. 아래는 `fixed-make`의 두 가지 예입니다. (함수 `quotient`은 두 수의 나눈 몫을, `remainder`는 나머지를 돌려줍니다.)

```
(define (fixed-make r d)
    (list r (quotient d 10)
            (remainder d 10)))

(define (fixed-make r d)
    (cons r d))
...
(define a (fixed-make 4 14))
(define b (fixed-make -7 0))
```

(b) 고정 소수점을 화면에 출력

이제 고정 소수점을 처리할 수 있는 함수들을 만들어서 제공해 봅시다. 한 개의 고정 소수점을 받아서 화면에 xx.xx 형식으로 출력해 주는 함수 `fixed-display`를 만들어봅시다. (화면에 출력할때는 `display`를 쓰면 되고, 여러 줄을 실행시키고 싶으면 `(begin (..) (..) (..))` 이런 식으로 `begin`을 사용하여 감싸면 됩니다)

```
(fixed-display a)
```

(c) 고정 소수점 수치 연산

고정소수점 더하기/빼기, 곱하기를 하는 함수 `fixed-add fixed-sub fixed-multiply`를 만들어 봅시다. 곱하기의 경우, 결과에서 소수점 3번째 자리에서 반올림한 값을 돌려줍니다.

```
(fixed-add a b)
(fixed-sub a b)
(fixed-multiply a b)
```

(d) 고점 소수점 비교 연산

두 개의 고정소수점을 받아서 두 값이 같으면 #t를 돌려주는 함수 `fixed-equal`을 만들어 봅시다.

```
(fixed-equal a b)
```

2. 2차원 고정 소수점 벡터 (순서쌍)

2차원 고정 소수점 벡터를 x좌표와 y좌표의 순서쌍으로 만들어 봅시다.

$$
\begin{aligned}
\texttt{make-vect2-pair} &: fixed \times fixed \to pair(fixed,\ fixed) \\
\texttt{nth-vect2-pair} &: pair(fixed,\ fixed) \times int \to fixed \\
\texttt{equal-vect2-pair} &: pair(fixed,\ fixed) \times pair(fixed,\ fixed) \to bool \\
\texttt{add-vect2-pair} &: pair(fixed,\ fixed) \times pair(fixed,\ fixed) \to pair(fixed,\ fixed) \\
\texttt{scale-vect2-pair} &: pair(fixed,\ fixed) \times fixed \to pair(fixed,\ fixed) \\
\texttt{dot-product-vect2-pair} &: pair(fixed,\ fixed) \times pair(fixed,\ fixed) \to fixed \\
\texttt{is-vect2-pair?} &: pair(fixed,\ fixed) \to bool
\end{aligned}
$$

```
(define x (fixed-make 5 34))
(define y (fixed-make -4 23))
(define z (fixed-make 2 42))
(define p1 (make-vect2-pair x y))
(define p2 (make-vect2-pair y z))

(fixed-display (nth-vect2-pair p1 0))
5.34
(fixed-display (nth-vect2-pair p2 1))
2.42
(equal-vect2-pair p1 p2)
#f
(equal-vect2-pair p1 p1)
#t
(define p3 (scale-vect2-pair (add-vect2-pair p1 p2) z))
(fixed-display (dot-product-vect2-pair p1 p3))
32.62
(is-vect2-pair? x)
#f
(is-vect2-pair? p3)
#t
```

`make-vect2-pair` 는 x좌표와 y좌표를 받아 벡터를 만듭니다. `nth-vect2-pair`는 만들어진 벡터와 정수를 받아 해당하는 좌표값을 돌려줍니다. 0이면 x좌표, 1이면 y좌표를 돌려주면 됩니다. `equal-vect2-pair`는 두 좌표가 모두 같을 때에만 같다고 정의됩니다. 나머지 함수들도 일반적인 벡터연산처럼 동작하게 하면 됩니다.

3. 고정 소수점 벡터

n차원 고정 소수점 벡터 타입 *vect*를 만들어 봅시다. 가장 간단하게는 길이 n인 고정 소수점 리스트로 구현할 수 있습니다. 1번에서 정의된 것과 마찬가지 동작을 하면 됩니다.

$$\begin{aligned}
\texttt{make-vect} &: fixed\ list \rightarrow vect \\
\texttt{nth-vect} &: vect \times int \rightarrow fixed \\
\texttt{equal-vect} &: vect \times vect \rightarrow bool \\
\texttt{add-vect} &: vect \times vect \rightarrow vect \\
\texttt{scale-vect} &: vect \times fixed \rightarrow vect \\
\texttt{dot-product-vect} &: vect \times vect \rightarrow fixed \\
\texttt{is-vect?} &: vect \rightarrow bool
\end{aligned}$$

```
(define v1 (make-vect (list x y z x)))
(define v2 (make-vect (list x y x z)))

(fixed-display (nth-vect v1 2))
2.42
(fixed-display (nth-vect v2 3))
2.42
(equal-vect v1 v2)
#f
(equal-vect v1 v1)
#t
(define v3 (scale-vect (add-vect v1 v2) z))
(fixed-display (dot-product-vect v1 v3))
352.50
(is-vect? x)
#f
(is-vect? v3)
#t
```