Principles of Programming, Fall 2011

Practice 3

User defined type, Implementation of data in various methods

Programming Research Lab,@SNU
Seungjung Lee, Youngseok Lee

October 10, 2011

The Objectives of this practice:

- Indicate the type of data and use it.
- Program the implementation of data.
- Program in various ways of the implementation of data.
- Learn to program only with the interface of data without knowing the implementation

WARNING Let get into a habit of keeping in mind of the types while programming the exercise problems. Also make sure you handle the error correctly for wrong parameter input, all the time.

Exercise

1. User defined type - fixed point

   Let's make a fixed point types that can express numbers to the hundredth only. Fixed point express real number in its integer part and decimal part separately. Since we can't express the decimals indefinitely, we restrain its size. Assume that the decimal parts are of integer from 0 to 99. in case of negative numbers, adjust the integer part of the number so that the decimals are of integer from 0 to 99.

   We need storage to save two integer to express the fixed point. There can be lots of ways to save two integers. For example, 3.04 can be expressed as a pair `(cons 3 4)`, or as a list `'(3 0 4)`. It can also be expressed as a list of pairs that has indications to

each number such as `(list (cons 'r 3) (cons 'd2 4))`. It's up to you to decide the inner parts.

(a) Making a fixed point

Let's make a function `fixed-make` that receives two numbers, a integer part and a fixed decimal part, and returns the value in the fixed point type. The following is two examples of `fixed-make`. (The function `quotient` returns the quotient of the two numbers, and `remainder` function returns the remainder of the two numbers.)

```
(define (fixed-make r d)
    (list r (quotient d 100)
            (quotient (remainder d 100) 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) Displaying the fixed point

Not, let's provide some functions that can deal with the fixed point. Define a functions `fixed-display` that displays a single received fixed point in form of xx.xx. (Just use `display` to print on the screen, and use `begin` in form of `(begin (..) (..) (..))` to run multiple lines.)

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

(c) Arithmetic operations of the fixed point

Let's make two functions `fixed-add fixed-multiply`. In case of multiply, return the value which is rounded off to three decimal places.

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

(d) Comparison of the fixed point

Implement a function `fixed-equal` that returns #t if two given fixed point are equal.

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

2. 2-dimensional fixed point vector (pair)

   Convert the 2-dimensional fixed point vector into a pair of x-coordinate and y-coordinate.

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

   `make-vect2-pair` receives x-coordinate and y-coordinate and produces a vector. `nth-vect2-pair` receives the vector and an integer and returns the appropriate coordinate. It returns the x-coordinate for 0, and y-coordinate for 1. `eqaul-vect2-pair` consider equal only when both coordinates are equal. Implement other functions so that it works as normal vector operations.

3. Fixed point vector

   Implement a n-dimensional fixed point vector type *vect*. It can simply be implemented by a fixed point list of lennth n as long as it works the same way as the one defined in exercise 1. Be careful for undefined inputs.

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

4. 2-dimensional fixed point vector implemented in two different methods

   We have learned two ways to make the 2-dimensional fixed point vector from exercise 2 and 3. Implement the following functions that works just fine for and 2-dimensional fixed point vector makde from either of the two methods. Define the return of a vector as the return of a pair. Be aware of undefined inputs.

`nth-vect2`	:	$vect2 \times int \rightarrow fixed$
`equal-vect2`	:	$vect2 \times vect2 \rightarrow bool$
`add-vect2`	:	$vect2 \times vect2 \rightarrow vect2$
`scale-vect2`	:	$vect2 \times fixed \rightarrow vect2$
`dot-product-vect2`	:	$vect2 \times vect2 \rightarrow fixed$