Homework 1 - Higher-order procedures and symbols

高阶函数 highorder.scm

所谓高阶函数就是以函数为参数或者以函数为返回值的函数。

你需要依次实现五个非常常见的高阶函数，不允许使用 scheme 自带的高阶函数。

1. curring

curring 也称柯里化，参考 wikipedia。

curry 函数的类型为 `((a, b) -> c) -> a -> b -> c`，含义为它可以把一个 "接受 a 类型和 b 类型参数并返回 c 类型的函数" 变成一个 "接受 a 类型的参数并返回 '接受 b 类型的参数并返回 c 类型的函数' 的函数"。

这个套娃的实际意义在于告诉我们所有多参数的函数都可以被转换为单个参数的函数的组合。

```
; 1. curring :: ((a, b) -> c) -> a -> b -> c
(define (curry f)
    'your-code-here)
```

2. compose

compose 为函数复合，类型为 `(b -> c) -> (a -> b) -> (a -> c)`，`(compose f g)` 的含义为 $f \circ g$。

```
; 2. compose :: (b -> c) -> (a -> b) -> (a -> c)
(define (f x)
  'your-code-here)
```

3. map

map 函数的类型为 `(a -> b) -> [a] -> [b]`，含义为接受一个把 a 类型数据映射到 b 类型数据的函数，并把该函数逐个应用于列表中的每个元素，得到一个 b 类型构成的列表。

```
; 3. map :: (a -> b) -> [a] -> [b]
; e.g. map(f, [x, y, z]) = [f(x), f(y), f(z)]
(define (map f xs)
    'your-code-here)
```

4. foldr

foldr 函数的类型为 `((a, b) -> b) -> b -> [a] -> b`，含义为从右往左把列表变成折叠成一个值。

比如 `foldr(+ 0 (list x y z))` 求的就是 `x + (y + (z + 0))`。

```
; 4. foldr :: ((a, b) -> b) -> b -> [a] -> b
; e.g. foldr(f, init, [x, y, z]) = f(x, f(y, f(z, init)))
(define (foldr f init xs)
    'your-code-here)
```

5. filter

filter 函数的类型为 `(a -> Bool) -> [a] -> [a]`，意思是接受一个把 a 类型数据映射到布尔类型数据的函数，并把使该函数为真的那些元素保留下来。

```
; 5. filter :: (a -> Bool) -> [a] -> [a]
(define (filter f xs)
    'your-code-here)
```

多项式求导器 deriv.scm

你需要实现一个为多项式求导的函数 `(deriv wrt expr)`，表示求 `expr` 这个表达式对 `wrt` 求导。

代码提供了一些抽象类型的构造器、访问器以及判断类型的谓词，无需更改。

```
(define op-add '+)
(define op-mul '*)

; Constructors
(define (make-binary-expr op lhs rhs) (list op lhs rhs))
(define (make-add-expr lhs rhs) (make-binary-expr op-add lhs rhs))
(define (make-mul-expr lhs rhs) (make-binary-expr op-mul lhs rhs))

; Accessors
(define (get-op expr) (car expr))
(define (get-lhs expr) (cadr expr))
(define (get-rhs expr) (caddr expr))

; Predicates
(define constant? number?)
(define variable? symbol?)
(define same-variable? equal?)
(define (add-expr? expr) (equal? (get-op expr) op-add))
(define (mul-expr? expr) (equal? (get-op expr) op-mul))
```

你需要实现：

1. 常数求导

$$\frac{\mathrm{d}c}{\mathrm{d}x} = 0$$

2. 变量求导

$$\frac{\mathrm{d}x}{\mathrm{d}x} = 1$$

3. 加法求导

$$\frac{\mathrm{d}(u+v)}{\mathrm{d}x} = \frac{\mathrm{d}u}{\mathrm{d}x} + \frac{\mathrm{d}v}{\mathrm{d}x}$$

4. 乘法求导

$$\frac{\mathrm{d}(uv)}{\mathrm{d}x} = u\frac{\mathrm{d}v}{\mathrm{d}x} + v\frac{\mathrm{d}u}{\mathrm{d}x}$$

最后将这四者组合起来就可以得到对任意仅包含加法乘法的多项式求导器了。

为了验证结果的方便，你**无需化简结果**，链式法则的顺序必须与上式**一致**。

```
(define (deriv-constant wrt constant)
    'your-code-here)

(define (deriv-variable wrt var)
    'your-code-here)

(define (deriv-sum wrt expr)
    'your-code-here)

(define (deriv-mul wrt expr)
    'your-code-here)

; put deriv-constant, deriv-variable, deriv-sum and deriv-mul together
(define (deriv wrt expr)
    (cond ; your-code-here
          (else (error "Don't know how to differentiate.\n" expr))))
```

参考资料

1. D. Friedman, M. Felleisen, *The little Schemer (4th ed.)*. MIT Press, 1996. [pdf]
2. H. Abelson, G. Sussman, *Structure and Interpretation of Computer Programs*. MIT Press, 1996. [pdf]