



*Problem 4*

```
(define team-name car)
(define team-wins cadr)
(define team-ties caddr)
(define team-losses caddr)

or

(define (team-name team) (car team))
(define (team-wins team) (cadr team))
(define (team-ties team) (caddr team))
(define (team-losses team) (caddr team))

(define first-team car)
(define rest-teams cdr)

or

(define (first-team teams-list) (car teams-list))
(define (rest-teams teams-list) (cdr teams-list))

(define (winning-teams teams-list)
  (cond ((null? teams-list) nil)
        ((> (team-wins (first-team teams-list))
             (team-losses (first-team teams-list)))
         (cons (first-team teams-list)
               (winning-teams (rest-teams teams-list))))
        (else (winning-teams (rest-teams teams-list)))))
```

*Problem 5*

```
(define (min-of-f-x-and-g-x f g x)
  (min (f x)
       (g x)))

(define (combine-f-x-and-g-x combiner f g x)
  (combiner (f x)
            (g x)))

(define (mul-f-x-and-g-x f g x)
  (combine-f-x-and-g-x * f g x))
```

*Problem 6*

```
(define (add-two-lists lst1 lst2)
  (if (null? lst1)
      nil
      (cons (+ (car lst1) (car lst2))
            (add-two-lists (cdr lst1) (cdr lst2)))))
```

Time:  $\Theta(n)$

Space:  $\Theta(n)$

Recursive process