

## ASSIGNMENT #6 SOLUTION

```
1. redstone% cat perfect.lsp
; perfect returns the list of factors of n if n is a perfect number, and nil
; otherwise

(defun perfect (n)
  (prog (nfactors)
    (cond
      ((= (mod n 2) 0) (setq nfactors (factors n (/ n 2))))
      (t (setq nfactors (factors n (/ (- n 1) 2))))
    )
    (cond
      ((= n (sumlist nfactors)) (return nfactors))
      (t nil)
    )
  )
)

; factors is a function that returns all of the factors of n up to m.

(defun factors (n m)
  (cond
    ((= m 1) (list 1))
    ((= (mod n m) 0) (append (factors n (- m 1)) (list m)))
    (t (factors n (- m 1)))
  )
)

; sumlist computes the sum of a list of numbers.

(defun sumlist (numberlist)
  (eval (cons '+ numberlist)))

redstone% gcl
GCL (GNU Common Lisp) Version(2.2) Thu Mar 28 13:41:41 CST 1996
Licensed under GNU Public Library License
Contains Enhancements by W. Schelter
Loading init.lsp
Finished loading init.lsp

>(load "perfect")
Loading perfect.lsp
Finished loading perfect.lsp
T

>(perfect 6)
(1 2 3)

>(perfect 298)
NIL

>(perfect 496)
(1 2 4 8 16 31 62 124 248)
```

```

redstone% cat drawcard.lsp
(load "carddeck")

; drawcard selects a card at random from a deck of cards which has not been
; drawn previously, as indicated by being in cardlist, and returns the value
; of that card

(defun drawcard (cardlist)
  (nth (drawcardnumber cardlist) (carddeck)))

; drawcardnumber selects a card at random from a deck of cards which has not
; been drawn previously, as indicated by being in cardlist, and returns the
; index of that card in the card deck.

(defun drawcardnumber (cardlist)
  (prog (cardnumber)
    (setq cardnumber (random 52))
    (cond
      ((member cardnumber cardlist) (return (drawcardnumber cardlist)))
      (t (return cardnumber)))
    )
  )
)

; member determines if an integer n appears in a list or not.

(defun member (n list)
  (cond
    ((null list) nil)
    ((= n (car list)) 't)
    (t (member n (cdr list))))
  )
)

redstone% gcl
GCL (GNU Common Lisp) Version(2.2) Thu Mar 28 13:41:41 CST 1996
Licensed under GNU Public Library License
Contains Enhancements by W. Schelter
Loading init.lsp
Finished loading init.lsp

>(load "drawcard")
Loading drawcard.lsp
Loading carddeck.lsp
Finished loading carddeck.lsp
Warning: MEMBER is being redefined.
Finished loading drawcard.lsp
T

>(drawcard nil)
(JACK HEART)

```

```

>(drawcard '(23))
(2 CLUB)

>(drawcard '(23 40))
(2 HEART)

>~D
redstone% cat blackjack.lsp
(load "drawcard")

; blackjack simulates a game of blackjack

(defun blackjack ()
  (playblackjack nil nil nil))

; playblackjack plays a game of blackjack by making turns with the player and
; dealer until the game is over.

(defun playblackjack (playercards dealercards drawncards)
  (cond
    ((eq (car (last playercards)) 'bust) ; if player busts, dealer wins
      (finalize playercards 'lose dealercards 'win))
    ((eq (car (last dealercards)) 'bust) ; if dealer busts, player wins
      (finalize playercards 'win dealercards 'lose))
    ((and (eq (car (last playercards)) 'stand)
          (eq (car (last dealercards)) 'stand))
      (cond ; if both stand, most points win
        ((> (cardtotal playercards) (cardtotal dealercards))
          (finalize playercards 'win dealercards 'lose))
        ((< (cardtotal playercards) (cardtotal dealercards))
          (finalize playercards 'lose dealercards 'win))
        (t (finalize playercards 'draw dealercards 'draw))
      )
    )
  )
  (t
    (prog (playerplay dealerplay)
      (setq playerplay (play playercards drawncards))
      (cond
        ((eq (car (last (car playerplay))) 'bust)
          (return ; if player busts, dealer stops
            (playblackjack (car playerplay) dealercards (cadr playerplay)))
        )
      )
      (setq dealerplay (play dealercards (cadr playerplay)))
      (return
        (playblackjack (car playerplay) (car dealerplay) (cadr dealerplay)))
      )
    )
  )
)
)
)
)
)

```

```

; finalize finalizes the result

(defun finalize (playercards playerresult dealercards dealerresult)
  (list
    (cons 'player (append playercards (list playerresult)))
    (cons 'dealer (append dealercards (list dealerresult))))
  )

; play calculates the value of the current set of cards and decides whether to
; take a hit or stand

(defun play (playercards drawncards)
  (cond
    ((eq (car (last playercards)) 'stand) (list playercards drawncards))
    ((>= (cardtotal playercards) 17)
     (list (append playercards (list 'stand)) drawncards))
    (t
     (prog (playerplay)
       (setq playerplay (deal playercards drawncards))
       (cond
         ((> (cardtotal (car playerplay)) 21)
          (return
           (list (append (car playerplay) (list 'bust)) (cadr playerplay))))
         (t (return playerplay)))
       )
     )
  )
)

; deal draws a card and adds it to the list of cards for the player and overall
; set of cards drawn.

(defun deal (playercards drawncards)
  (prog (cardnumber)
    (setq cardnumber (drawcardnumber drawncards))
    (return
     (list
      (append playercards (list (nth cardnumber (carddeck))))
      (append drawncards (list cardnumber)))
     )
  )
)

; cardtotal returns the total value of a list of cards. If the value of ace
; taken as 11 causes the total to exceed 21, the ace will be counted as 1.

(defun cardtotal (cardlist)
  (prog (total)
    (setq total (computecardtotal cardlist 11))
    (cond
      ((> total 21) (return (computecardtotal cardlist 1)))
      (t (return total))
    )
  )
)

```

```

    )
  )
)

; computecardtotal returns the total value of a list of cards using the
; designatedvalue of the ace.

(defun computecardtotal (cardlist acevalue)
  (cond
    ((null cardlist) 0)
    ((eq (car cardlist) 'stand) 0)
    (t (+ (cardvalue (car cardlist) acevalue)
          (computecardtotal (cdr cardlist) acevalue)))
  )
)

; cardvalue returns the value of a card.

(defun cardvalue (card acevalue)
  (cond
    ((eq (car card) 'ace) acevalue)
    ((eq (car card) 'jack) 10)
    ((eq (car card) 'queen) 10)
    ((eq (car card) 'king) 10)
    (t (car card))
  )
)

redstone% gcl
GCL (GNU Common Lisp) Version(2.2) Thu Mar 28 13:41:41 CST 1996
Licensed under GNU Public Library License
Contains Enhancements by W. Schelter
Loading init.lsp
Finished loading init.lsp

>(load "blackjack")
Loading blackjack.lsp
Loading drawcard.lsp
Loading carddeck.lsp
Finished loading carddeck.lsp
Warning: MEMBER is being redefined.
Finished loading drawcard.lsp
Finished loading blackjack.lsp
T

>(blackjack)
((PLAYER (JACK HEART) (2 HEART) (QUEEN DIAMOND) BUST LOSE)
 (DEALER (2 CLUB) (7 HEART) WIN))

>(blackjack)
((PLAYER (6 CLUB) (2 CLUB) (3 HEART) (5 HEART) (4 DIAMOND) STAND DRAW)
 (DEALER (2 DIAMOND) (7 HEART) (ACE HEART) STAND DRAW))

```

```
>(blackjack)
((PLAYER (ACE DIAMOND) (8 SPADE) STAND WIN)
 (DEALER (10 DIAMOND) (4 HEART) (8 HEART) BUST LOSE))
```

```
>~D
```

```
2. S1   input n;
   S2   count := n;
   S3   sum := 0;
   S4   while (count <> 0) loop
   S5       sum := sum + count;
   S6       count := count - 1;
           end loop;
   S7   output sum
```

$M[S_1; S_2; S_3; S_4; S_7] \langle 3, 2 \rangle$

$(S[S_1; S_2; S_3; S_4; S_7] \langle \lambda V.\perp, \langle 3, 2 \rangle, nil \rangle) \downarrow 3$

$(S[S_2; S_3; S_4; S_7] (S[\text{input } n] \langle \lambda V.\perp, \langle 3, 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_2; S_3; S_4; S_7] (\text{if } \langle 3, 2 \rangle = nil \text{ then } \top \text{ else } \langle \lambda V.\perp [hd(\langle 3, 2 \rangle)/n], tl \langle 3, 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_2; S_3; S_4; S_7] \langle \lambda V.\perp [hd(\langle 3, 2 \rangle)/n], \langle 2 \rangle, nil \rangle) \downarrow 3$

$(S[S_2; S_3; S_4; S_7] \langle \lambda V.\perp [3/n], \langle 2 \rangle, nil \rangle) \downarrow 3$

$(S[S_3; S_4; S_7] (S[\text{count} := n] \langle \lambda V.\perp [3/n], \langle 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_3; S_4; S_7] (\langle \lambda V.\perp [3/n] [E[n]] (\lambda V.\perp [3/n])/count], \langle 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_3; S_4; S_7] (\langle store_1 [\text{if } store_1 [n] = \perp \text{ then } \top \text{ else } store_1 [n] /count], \langle 2 \rangle, nil \rangle)) \downarrow 3$   
where  $store_1 = \lambda V.\perp [3/n]$

$(S[S_3; S_4; S_7] (\langle \lambda V.\perp [3/n] [3/count], \langle 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_4; S_7] (S[\text{sum} := 0] \langle \lambda V.\perp [3/n] [3/count], \langle 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_4; S_7] (\langle store_2 [E[0] store_2] /sum], \langle 2 \rangle, nil \rangle)) \downarrow 3$ , where  $store_2 = \lambda V.\perp [3/n] [3/count]$

$(S[S_4; S_7] (\langle store_2 [N[0]] /sum], \langle 2 \rangle, nil \rangle)) \downarrow 3$ , where  $store_2 = \lambda V.\perp [3/n] [3/count]$

$(S[S_4; S_7] \langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle) \downarrow 3$

$(S[S_7] (S[\text{while } (count \neq 0) \text{ loop } S_5; S_6 \text{ end loop}] \langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_7]$

$(\text{if } C[\text{count} \neq 0] (\lambda V.\perp [3/n] [3/count] [0/sum]) \text{ then}$   
 $\frac{S[S_4] (S[S_5; S_6] \langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle)}{S[S_4] (S[S_5; S_6] \langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle)}$

else

$\langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_7]$

$(\text{if } E[\text{count}] (\lambda V.\perp [3/n] [3/count] [0/sum]) \neq E[0] (\lambda V.\perp [3/n] [3/count] [0/sum]) \text{ then}$   
 $\frac{S[S_4] (S[S_5; S_6] \langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle)}{S[S_4] (S[S_5; S_6] \langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle)}$

else

$\langle \lambda V.\perp [3/n] [3/count] [0/sum], \langle 2 \rangle, nil \rangle)) \downarrow 3$

$(S[S_7]$   
    (if  $3 \neq 0$  then  
         $\frac{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum], <2>, nil >)}{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum], <2>, nil >)}$   
    else  
         $< \lambda V. \perp [3/n][3/count][0/sum], <2>, nil >)) \downarrow 3$   
 $(S[S_7] (S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum], <2>, nil >))) \downarrow 3$   
 $(S[S_7] (S[S_4] (S[S_6] (\underline{S[sum := sum + count]} < \lambda V. \perp [3/n][3/count][0/sum], <2>, nil >)))) \downarrow 3$   
 $(S[S_7] (S[S_4] (S[S_6] (\underline{< \lambda V. \perp [3/n][3/count][0/sum][E[sum + count]] (\lambda V. \perp [3/n][3/count][0/sum])/sum}, <2>, nil >)))) \downarrow 3$   
 $(S[S_7] (S[S_4] (\underline{S[count := count - 1]} (\lambda V. \perp [3/n][3/count][0/sum][3/sum], <2>, nil >)))) \downarrow 3$   
 $(S[S_7] (S[S_4] (\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count], <2>, nil >))) \downarrow 3$   
 $(S[S_7]$   
    (if  $C[count <> 0]$   $(\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count])$  then  
         $\frac{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count], <2>, nil >)}{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count], <2>, nil >)}$   
    else  
         $< \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count], <2>, nil >)) \downarrow 3$   
 $(S[S_7] (S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count], <2>, nil >))) \downarrow 3$   
 $(S[S_7] (S[S_4] (\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count], <2>, nil >))) \downarrow 3$   
 $(S[S_7]$   
    (if  $C[count <> 0]$   $(\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count])$  then  
         $\frac{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count], <2>, nil >)}{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count], <2>, nil >)}$   
    else  
         $< \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count], <2>, nil >)) \downarrow 3$   
 $(S[S_7] (S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count], <2>, nil >))) \downarrow 3$   
 $(S[S_7] (S[S_4] (\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], <2>, nil >))) \downarrow 3$   
  
 $(S[S_7]$   
    (if  $C[count <> 0]$   $(\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count])$  then  
         $\frac{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], <2>, nil >)}{S[S_4] (S[S_5; S_6] < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], <2>, nil >)}$   
    else  
         $< \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], <2>, nil >)) \downarrow 3$   
 $(\underline{S[output sum]} < \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], <2>, nil >)$   
 $\downarrow 3$   
    (if  $(\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count])[\underline{sum}] = \perp$  then  
         $\perp$   
    else  
         $< \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], <2>,$   
         $\underline{append nil (list (\lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count])[\underline{sum}]))} >)$   
 $\downarrow 3$   
    (if  $\underline{6} = \perp$  then  
         $\perp$   
    else  
         $< \lambda V. \perp [3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], <2>, \underline{append nil (list 6)} >)$   
 $\downarrow 3$

( $\langle \lambda V.\perp[3/n][3/count][0/sum][3/sum][2/count][5/sum][1/count][6/sum][0/count], \langle 2 \rangle, \langle 6 \rangle \rangle \downarrow 3$   
 $\langle 6 \rangle$