Problem 4.

When the simulated games in the previous Problem ran, it was impossible for us to tell what was going on. It would be nice if we could watch a strategy play by observing its inputs and the decisions it makes. Define a procedure called **watch-player** that takes a strategy as an argument and returns a strategy as its result. The strategy returned by **watch-player** should implement the same result as the strategy that was passed to it as an argument, but, in addition, it should print the information supplied to the strategy and the decision that the strategy returns. For example,

```
(test-strategy (watch-player (stop-at 16))
               (watch-player (stop-at 15))
               2)
```

should play two simulated games and show what each player does at each step. Turn in a listing of your procedure and some sample runs using it.

Answer.

As is described in the problem, the procedure **watch-player** takes a strategy as argument and return a strategy as its value. Remember that a strategy in our implementation is represented as a procedure of two arguments: the player's hand and the point value of the opponent's face-up card. Thus, we can immediately draw a profile of the procedure **watch-player**:

```
(define (watch-player strategy)
  (lambda (my-hand opponent-up-card)
    (...)))
```

Further more, as the strategy returned by **watch-player** should implement the same result as itself originally. So, the body of **watch-player** contains a conditional, which is much similar to the case we dealt with in problem 2.

```
(cond ((strategy my-hand opponent-up-card)
       #t)
      (else #f))
```

So far, we've design the logical module of our **watch-player** procedure. To complete our work, we have to add in the interactive module. Well, that's easy! The interative module here resembles to the procedure **hit?** we've seen in the problem description a lot and all we have to do is integrating it with the logical module. Therefore, the complete version of our **watch-player** procedure turns out to be:

```
(define (watch-player strategy)
  (lambda (my-hand opponent-up-card)
    (newline)
    (newline)
    (display "Opponent up card ")
    (display opponent-up-card)
    (newline)
    (display "Your Total ")
    (display (hand-total my-hand))
    (newline)
    (display "Hit? ")
    (cond ((strategy my-hand opponent-up-card) ; hit?
           (display "y")
           #t)
          (else (display "n")
                #f))))        ; stay
```