Problem 2.

Define a procedure `stop-at` that takes a number as argument and returns a strategy procedure. The strategy `stop-at` should ask for a new card if and only if the total of a hand less than the argument to `stop-at`. For example `(stop-at 16)` should return a strategy that asks for another card if the hand total is less than 16, but stops as soon as the total reaches 16. To test your implementation of `stop-at`, play a few games by evaluating

```
(twenty-one hit? (stop-at 16))
```

Thus, you will be playing against a house whose strategy is to stop at 16. Turn in a listing of your procedure.

Answer.

As is mentioned above, `stop-at` returns a strategy procedure. And since we have been informed in the project description that a strategy 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 `stop-at`:

```
(define (stop-at n)
  (lambda (my-hand opponent-up-card)
    (...)))
```

On the other hand, we have known that `stop-at` would ask for a new card whenever the total of a hand less than its argument. Undoubtfully, the case "the total of a hand less than the argument to `stop-at`" can be expressed as:

```
(< (hand-total my-hand) n)
```

The real problem here is how to represent the request "ask for a card" in our strategy procedure? Well, remember that the project description had also made it quite clear: the procedure returns true if the player would want another card, and false if the player would stay. So, the body of our `stop-at` procedure is therefore a conditional

```
(if (< (hand-total my-hand) n)
    #t
    #f)
```

Combining the building blocks we have crafted just now, the panorama of the procedure `stop-at` turns out to be:

```
(define (stop-at n)
  (lambda (my-hand opponent-up-card)
    (if (< (hand-total my-hand) n)
        #t
        #f)))
```