CS3 Midterm Problem (sp06 midterm 1): Sub-cursion?
Standards and Solutions

Write the procedure `sub-sentence`, which returns a middle section of a sentence. It takes three parameters; the first identifies the index to start the middle section, and will be 1 or greater; the second identifies the length of the middle section, and will be 0 or greater; and the last is the sentence to work with.
Do *not* use any helper procedures.
Do *not* use the `item` procedure in your solution.

```
(sub-sentence 2 3 '(a b c d e f g)) ➔ (b c d)
(sub-sentence 3 2 '(a b)) ➔ ()
(sub-sentence 3 0 '(a b c d e) ➔ ()
(sub-sentence 3 9 '(a b c d e) ➔ (c d e)
```

This is an accumulating recursion: in some fashion, you need to count down *two* different times: once to get to the starting position and another time to get to the end of the sub-sentence (i.e., counting down length). For the later recursive call, you will need to be collecting up the sentence.

```
(define (sub-sentence start len sent)
   (cond ((empty? sent)
          ;; stop the recursion if our sentence ends,
          ;;  no matter what has happened yet
          '())
         ((> start 1)
          ;; we need to get to the starting position
          (sub-sentence (- start 1) len (bf sent))
         ((> len 0)
          ;; we've made it to the starting position, and now we
          ;; need to gather up the sub-sentence while moving
          ;; to the end of the sentence
          (se (first sent)
              (sub-sentence start (- len 1) (bf sent))))
         (else
          ;; we're at the end of the sub-sentence (len=0), so
          ;; we can just stop now.
          '())
         ))
```

Most of you put the last base case (where `len` is 0) at the top, but notice that it works fine after the recursive cases! The else clause also catches the situation where the start argument is greater then the `count` of `sent`. It also catches the situation where the `(+ len start)` goes beyond the `count` of `sent`.

A very few of you changed the second recursive call to lose words off the end of the sentence, and changed the base case to return the current sentence when the length of that sentence was equal to (or less than) the length of the sentence you were to return. This was a neat solution.

Some common errors include:

- Bad placeholder names
- Forgetting to check for an empty list
- Trying to remove elements from the end of the list (it is possible, but it is more difficult