

1. (1.5 points) Scheme Primer (Conceptual)

(a) Describe all interpretations of Scheme parentheses that you can think of (in other words, say you see some parentheses... what could their meaning be?).

Parentheses either denote procedure calls or special forms. Importantly, note that every set of parentheses counts; you can never leave them out and you can never add more.

(b) On a scale from 1 to 10, where 1 is “not at all” and 10 is “more than anything in the world,” how much do you like counting parentheses? Select one:

- 10

(c) What is a symbol in Scheme?

Symbols are like code itself – specifically symbols are immutable, interned strings. You can think of them as variable names; in this way symbols will come in handy where interpreters are concerned!

2. (2 points) WWSP?

```
scm> '(list 2 3)
(list 2 3)
```

```
scm> (list '(2 3))
((2 3))
```

```
scm> (define x (+))
x
scm> (define y +)
y
scm> (x 3 4)
Error: cannot call: 0
```

```
scm> (y 3 4)
7
```

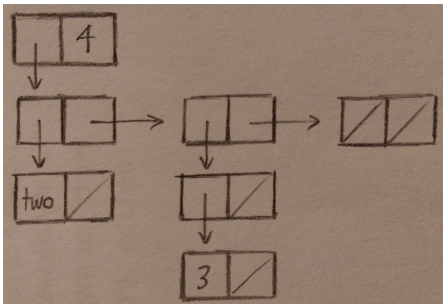
3. (2.5 points) Box and Pointers

Draw box-and-pointer diagrams for each of the following Scheme lists.

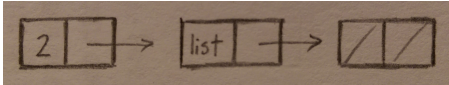
```
scm> '(2 . 3 4)
```

Error; you can only have a single element after a dot!

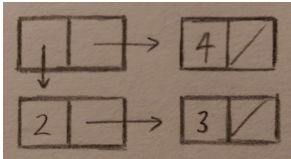
```
scm> (cons (list '(two) '((3)) nil) 4)
```



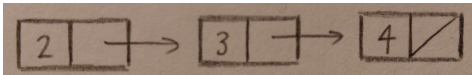
```
scm> (cons 2 '(list nil))
```



```
scm> (list (append '(2) '(3) nil) 4)
```



```
scm> '(2 . (3 . (4)))
```



4. (4 points) Last One

Implement the procedure `finish-sort`, which takes in a well-formed list `lst` (of distinct real numbers) and returns its sorted form. You can assume that `lst` is almost sorted already, such that exactly one number is somewhere to the **right** of where it belongs and everything else is in its relatively proper place. Thus it is possible to sort the list by shifting a **single** element to the left. To balance out this relaxation, `finish-sort` is only allowed to make one pass over the data, i.e. at most one recursive call per position in the list.

Hint: you may find both the `append` procedure and the `let` special form helpful.

```
(define (finish-sort lst)
  (define (cadr lst) (car (cdr lst)))
  (define (caddr lst) (car (cddr lst)))
  (cond
    ((or (null? lst) (null? (cdr lst))) lst)
    ((> (car lst) (cadr lst)) (append (list (cadr lst)) (list (car lst)) (caddr lst)))
    (else (let ((rest (finish-sort (cdr lst))))
              (if (< (car lst) (car rest))
                  (cons (car lst) rest)
                  (append (list (car rest)) (list (car lst)) (cdr rest))))))
  )
)
```

Example usage:

```
scm> (finish-sort '(2 3 4 5 6 7 1))
(1 2 3 4 5 6 7)
scm> (finish-sort '(2 1))
(1 2)
scm> (finish-sort '(2 9 3 11))
(2 3 9 11)
```