MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 2006

Quiz I

Closed Book – one sheet of notes

Throughout this quiz, we have set aside space in which you should write your answers. Please try to put all of your answers in the designated spaces, as we will look only in this spaces when grading.

Note that any procedures or code fragments that you write will be judged not only on correct function, but also on clarity and good programming practice. Also note that while there may be a lot of reading to do on a problem, there is relatively little code to write, so please take the time to read each problem carefully.

NAME:

Section Time: Tutor's Name:

PART	Value	Grade	Grader
1	20		
2	15		
3	20		
4	25		
5	20		
Total	100		

For your reference, the TAs are:

- Vikki Chou
- Tom Lasko
- Alex Vandiver

Part 1: (20 points)

For each of the following expressions or sequences of expressions, state the value returned as the result of evaluating the final expression in each set, after evaluating any previous expressions in the set; or indicate that the evaluation results in an error. If the result is an error, state in general terms what kind of error (e.g. you might write "error: wrong type of argument to procedure"). If the evaluation returns a built-in procedure, write `primitive procedure`. If the evaluation returns a user-created procedure, write `compound procedure`.

For each question, we also ask you to identify the "type" of the returned expression, using the notation introduced in lecture (assuming that the expression does not result in an error).

You may assume that evaluation of each sequence takes place in a newly initialized Scheme system.

Question 1.

```
(lambda (a b) (+ 1 a))
```

Value: Type:

Question 2.

```
((lambda (x y)
    (/ (- x y)
       (+ x y)))
 6 3)
```

Value: Type:

Question 3.

```
((lambda (a + b)
    (+ a b))
 3 * 4)
```

Value: Type:

Question 4.

```
((lambda (x)
     (lambda (y)
        (- (* x x) (* y y))))
  5)
```

Value: Type:

Question 5.

```
+
```

Value: Type:

In the rest of this quiz, we are going to be exploring a simple system for keeping tracking of voting results, for use in the upcoming election. The basic element of the system will be a record of votes for a candidate in a specific precinct (or voting area).

While the questions all follow a common theme and build on one another, they are all independent: Even if you get a question wrong, or leave it blank, you can still go on to the rest of the questions and answer them. In those later questions you can use the names of the functions you were asked to write in earlier questions, whether or not you got that previous question correct.

Part 2: (15 points)

First we need to worry about a data structure for representing this information. We will assume that we have a **constructor** called `precinct-data` which takes as arguments a name of a candidate (represented as a string), a number of votes for that candidate, and the number of the precinct, in that order. For example

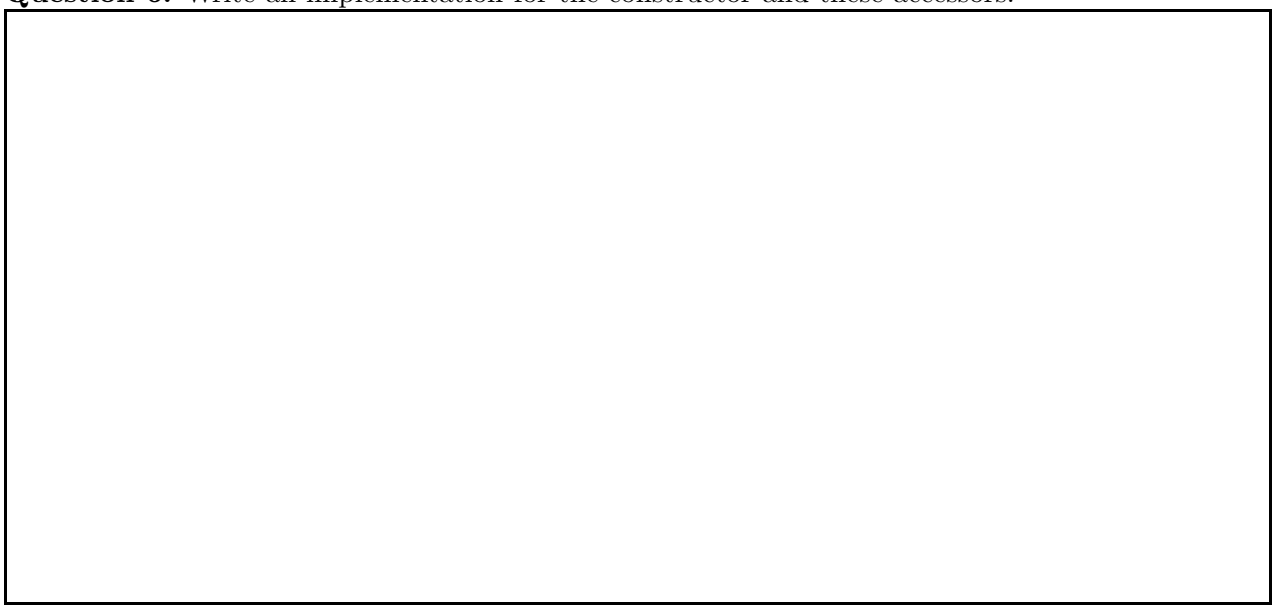
```
(define data1 (precinct-data "fred" 203 1))
(define data2 (precinct-data "judy" 253 1))
(define data3 (precinct-data "fred" 102 2))
(define data4 (precinct-data "judy" 203 2))
(define data5 (precinct-data "fred" 193 3))
(define data6 (precinct-data "judy" 143 3))

(define test-votes (list data1 data2))  ; simple list case

(define allvotes (list data1 data2 data3 data4 data5 data6)) ;; larger test case
```

Associated with this constructor are several **selectors** or **accessors**: `who, votes` and `precinct` each extract the associated parts of a precinct count.

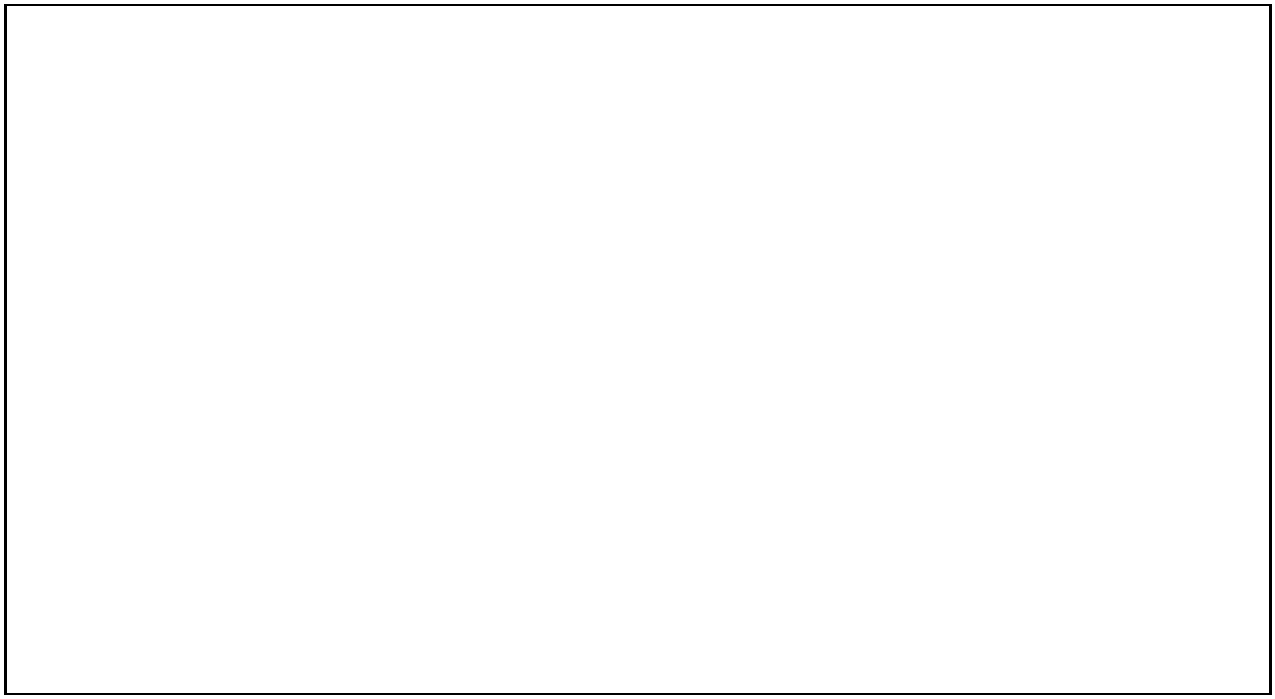
Question 6. Write an implementation for the constructor and these accessors:



Question 7. In the space provided, draw a box-and-pointer diagram for your implementation for the structure bound to `test-votes`, which is defined on the previous page.

```
(define data1 (precinct-data "fred" 203 1))
(define data2 (precinct-data "judy" 253 1))

(define test-votes (list data1 data2))  ; simple list case
```



Part 3: (20 points)

Assume that the procedure `addup` exists, with the following behavior: it takes as argument a list of numbers, and it returns as value the sum of those numbers.

With that assumption, we want to write a procedure, called `get-votes-for`, which takes as arguments a name of a candidate (as a string) and a list of precinct data objects (such as the example shown for `allvotes`). It returns as value the sum of the votes from all precincts for the candidate. We have provided a template below:

```
(define (get-votes-for cand data)
  (addup
    (map ANSWER8
         (filter ANSWER9 data))))
```

where

```
(define (map proc lst)
  (if (null? lst)
      '()
      (cons (proc (car lst))
            (map proc (cdr lst)))))

(define (filter pred lst)
  (cond ((null? lst) '())
        ((pred (car lst)) (cons (car lst) (filter pred (cdr lst))))
        (else (filter pred (cdr lst)))))
```

Question 8. Complete the code needed for ANSWER8

Question 9. Complete the code needed for ANSWER9 (you may find it useful to use `string=?`)

Question 10. Now write an implementation of `addup` that has a linear recursive process.

Question 11. As an alternative, write an implementation of `addup` that has an interative process.

Part 4: (25 points)

If our election has a lot of precincts (which it certainly will have in reality) then the list of precincts (such as `allvotes`) will be very long. So we might want to sort this list into a new list, where the entries are sorted, for example by precinct number.

Here are a pair of procedures that sort a list:

```
(define (sort lst comp)
  (define best-and-rest (bubble-up lst comp))
  (if (null? (cdr best-and-rest))
      best-and-rest
      (cons (car best-and-rest)
            (sort (cdr best-and-rest) comp))))

(define (bubble-up lst comp)
  (define (help best done todo)
    (cond ((null? todo)
           (cons best done))
          ((comp best (car todo))
           (help best (cons (car todo) done) (cdr todo)))
          (else (help (car todo) (cons best done) (cdr todo)))))
  (help (car lst) '() (cdr lst)))
```

Question 12: To help you absorb the code above, what is the value of

```
(bubble-up (list 2 1 3) <)
```

What is the value of

```
(sort (list 2 1 3) <)
```

Question 13: Using these procedures, complete the following procedure to sort a list of precinct data by precinct number, in increasing order.

```
(define (sort-by-precinct data)
  (sort data ANSWER7))
```

Question 14: What is the order of growth in time of the procedure `bubble-up` measured in terms of the length of the argument list?

Choose from:

A: constant
B: linear
C: exponential
D: quadratic
E: logarithmic
F: something else

What is the order of growth in space, measured as the maximum number of deferred operations, also measured as a function of the length of the argument list?

Question 15: What is the order of growth in time of the procedure `sort` measured in terms of the length of the argument list?

Choose from the same list of options:

What is the order of growth in space, measured as the maximum number of deferred operations, also measured as a function of the length of the argument list?

Part 5: (20 points)

While the code described above lets us sort precinct data, in a variety of ways, it is a bit cumbersome as it includes some redundant data. Suppose, instead, that we want to store vote information in a new format. This new format, called a **candidate record** includes the name of the candidate, plus a list of **precinct records** each of which is a combination of number of votes and precinct number. In other words, this new format has the candidate's name included only once.

Here is an implementation of such an abstraction:

```
(define (make-candidate-record name precinct-records)
  (list name precinct-records))

(define (make-precinct-record votes precinct)
  (list votes precinct))
```

We want you to write a procedure that will convert data from our original format, into this new format, for a specific candidate. Here is the main procedure:

```
(define (convert-form data candidate)
  (make-candidate-record candidate (construct-votes data candidate))
```

so that for example evaluating

```
(convert-form allvotes "fred")
```

returns the value

```
("fred" ((203 1) (102 2) (193 3)))
```

Question 16:

Write the procedure `construct-votes`:

Question 17: The procedure we just wrote will convert the precinct records for a specific candidate to the new form. Suppose instead we want to convert all the records. We can do this by using `convert-form`, so long as we provide the right set of data to the procedure.

Below is a template of code to do this:

```
(define (convert-all-new data)
  (if (null? data)
      '()
      ANSWER17
  )))
```

Provide the coded needed for ANSWER17. Hint: you may want to use `filter` to extract portions of the `data` to process, although you may also write procedures to do this directly.