NAME:____________________ **SCORE:_______________**
LOGIN: ___________________

CS61A Midterm #1 L. Rowe
Fall 2002

This examination is closed books, notes, and friends. Answer all questions in the space provided. If you do not understand a question, please ask the proctor for clarification.

Answer the questions using the Scheme functions discussed in class. You can use the word/sentence abstraction but not mutators (e.g., `set!`, etc.), vectors, or continuations. If you do not remember the specific arguments to a function or the order of the arguments to a function that takes several arguments, add a comment that describes how you are using the function. We are interested in the comments, not details about function interfaces.

Question	**Score**	**Total Possible**
1		(20 possible)
2		(10 possible)
3		(20 possible)
4		(10 possible)
5		(10 possible)
TOTAL		**(70 possible)**

Oath

I certify that I am the student whose name appears above.

Signature: ________________________________

Student ID #: _____________________________

Seating

On my left: **Name:____________________** **Login:______________________**

On my right: **Name:____________________** **Login:______________________**

1. (20 points) Show what Scheme will display or do if you enter each expression to the top-level interpreter (i.e., at the `STk>` prompt).

(i)
```
(list ‘list)
```

(ii)
```
(car (car (list ‘a)))
```

(iii)
```
(first ‘(word 1 2 3))
```

(iv)
```
((lambda () (/ 3 1))
```

(v)
```
(define (f) (g))
(define (g) f)
(f)
```

(vi)
```
((define (square x) (* x x)) 2)
```

(vii)
```
((lambda (x y) x) ‘(1 2))
```

(ix)
```
(cons (cons 1 ‘()) (cons 2 ‘()))
```

(x)
```
(let ((+ 0) (x +) (y 1))
   (x + y))
```

2. (10 points) This question builds on the Twenty-one game from project 1. Instead of representing a card by a word consisting of a number or letter and a letter for the suit, we are going to use a pair. The pair will include a number or letter for the rank of the card (i.e., 2, 3, …, 10, j, q, k, or a) and a letter for the suit (i.e., c, d, h, or s).

   For example the 4 of hearts and the king of diamonds will be represented as follows:

   ```
   (cons 4 ‘h)

   (cons ‘k ‘d)
   ```

   (i) (2 points) Write a function `make-card` that takes a rank and suit and returns a pair representing the card.

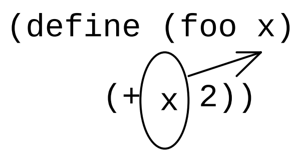
   (ii) (4 points) Write two accessor functions `rank` and `suit`. `rank` takes a card and returns the number or letter of the card. `suit` takes a card and returns the suit. For example:

   ```
   (define four_of_hearts (make-card 4 ‘h))
   (define king_of_diamonds (make-card ‘k ‘d))

   (rank four_of_hearts)   => 4
   (rank king_of_diamonds) => k

   (suit four_of_hearts)   => h
   (suit king_of_diamonds) => d
   ```

   (iii) (4 points) Write the function `total` using the new representation of a card. `total` takes a “list of cards” and adds up their value assuming aces count as 11 and there are no jokers.

3. (20 points) A word is a doublet of another word if they differ in only one letter. For example, “noise” and “poise” are doublets, “poise” and “posse” are doublets, but “noise” and “posse” are not doublets because they differ in two letter. You are to write the procedure `doublet?` that takes two words and returns `#t` or `#f` indicating whether or not the two arguments are doublets. Doublet words must be the same length. (Hint: you may need a helper function although the problem can be solved without one.)

4. (10 points) Draw arrows in the following code to demonstrate which variables refer to which definitions. For example:



```
(define (foo x)
     (+ x 2))
```

shows that `x` in the body of the procedure is bound to the formal argument. Note that there is no binding for the formal argument, so no arrow is drawn from it.

(i) (5 points) Draw arrows to show the binding of each variable in the following examples:

```
(define (bar x)
   ((lambda (x) (* x x)) x))


(let ((x 3) (y 2))
  (define (func x) (+ x y))
  (let ((x 100) (y 200))
     (func x)))
```

(ii) (5 points) What does the `let` expression return?

5. (10 points) You are to write a procedure named `agrees?` that takes two arguments:

    1) a function `f` that takes one argument
    2) a list of pairs

The “list of pairs” will be possible values of computing `f`. In other words, the first value of the pair will be an actual argument and the second value will be a possible return result of the procedure. The `agrees?` procedure will determine whether `f` applied to the first value in each pair produces a result that matches the second value in the pair. You can think of this procedure as one the readers might construct to test a homework assignment. The `agrees?` procedure should return `#t` or `#f` depending on whether all pairs match. (Example on next page.)

For example,

```
(agrees? (lambda (x) (* x x))
     (list (cons 1 1) (cons 3 9))) => #t
(agrees? (lambda (x) (* x x))
     (list (cons 2 4) (cons 1 5) (cons 6 36))) => #f
(agrees? (lambda (x) (* x x)) ‘()) => #t
```

Write the function `agrees?`: