CS 61A Midterm #1 — Wed, Februrary 20th, 2002

Your name ________________________________

login: cs61a–______

Discussion section number ______

Circle your TA's name	Chris Cheng	Chris Karlof	David Schultz	Donald Chai	Erwin Vedar	Ryan Stejskal	Seema Moorjani	Todd Segal	Yaping Li

The student sitting to my right is ____________________

The student sitting to my left is ____________________

This exam is worth 50 points, or about 17% of your total course grade. It includes two parts: The individual exam (this part) is worth 44 points, and the group exam is worth 6 points. The individual exam contains six substantive questions, plus the following:

Question 0 (1 point): Fill out this front page correctly and put your name and login correctly at the top of each of the following pages.

This booklet contains six numbered pages including the cover page. Put all answers on these pages, please; don't hand in stray pieces of paper. This is an open book exam.

When writing procedures, don't put in error checks. Assume that you will be given arguments of the correct type.

Our expectation is that many of you will not complete one or two of these questions. If you find one question especially difficult, leave it for later; start with the ones you find easier.

READ AND SIGN THIS:

I certify that my answers to this exam are all my own work, and that I have not discussed the exam questions or answers with anyone prior to taking this exam.

If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

____

0	/1
1	/12
2–3	/7
4	/8
5	/8
6	/8
total	/44

Question 1 — I'm drawing a blank... (12 points):

(a) Fill in the *simplest possible answer* in the blank so that the expression produces the result.

```
> ((lambda (x) (x x x)) ________________________________________ )
usa
```

(b) Given the following two definitions, fill in zero or more parentheses only (PO) in each blank to complete the scheme expression and show the numeric return value on the right.

```
(define (double x) (* 2 x))

(define olympics
  (lambda (x)
    (lambda (f)
      (f (f x)))))

>__ olympics __ olympics __ 5 __ double __ double __ ==>  ________________
 PO          PO           PO  PO        PO        PO      must be a number
```

(c) Fill in the result. If the expression produces an error, just say "error"; if it returns a procedure, just say "procedure"; if it causes an infinite loop, just say "loop".

```
> (let ((y '(gold)))
    (cons (append y (list y)) y))
```

____

(d) Fill in the result. If the expression produces an error, just say "error"; if it returns a procedure, just say "procedure"; if it causes an infinite loop, just say "loop".

```
> (define (bronze n)
    (define (iter count i)
      (if (<= i 1)
          count
          (iter (+ count 1) (- i 2))))
    (iter 0 n))

> (bronze 5000)
```

____

(e) Fill in the result. If the expression produces an error, just say "error"; if it returns a procedure, just say "procedure"; if it causes an infinite loop, just say "loop". You may assume pigl is defined exactly as it is in the notes.

```
> (every pigl (se '() 'class '(beats) (word 61 'a)))
```

____

Question 2 — Eve Ensler was discussing "Big-O" notation... (3 points):

This question concerns the following functions.

```
(define (f x)        (define (g y)            (define (h z)
  (if (< x 0)          (if (< y 100)            (if (< z 0)
      1                    0                        0
      (f (- x 2))))        (+ (f y)                 (+ (h (- z 2))
                              (f (- y 1)))))           (h (- z 1)))))
```

Circle (T)rue or (F)alse:

T F h generates an iterative process (i.e., uses $\Theta(1)$ space).

T F g generates an iterative process (i.e., uses $\Theta(1)$ space).

T F f is $\Theta(x)$.

T F h is $\Theta(z^2)$.

T F f and g have the same order of growth.

T F g and h have the same order of growth.

Question 3 — Waiter, please take my order of evaluation... (4 points):

This question concerns the following functions.

```
(define (square x)
  (* x x))

(define (foo x y)
  (* x y y))

(define (bar x y)
  x)
```

(a) How many times is * called for `(foo (square (* 1 1)) (square (* 1 1)))`

...under normal order? ______________

...under applicative order? ______________

(b) How many times is * called for `(bar (square (* 1 1)) (square (* 1 1)))`

...under normal order? ______________

...under applicative order? ______________

Question 4 — If you can't be an athlete, be an athletic supporter... (8 points):

We designed an abstract data type, `athlete`, to represent Olympic athletes. An `athlete` contains 3 pieces of information: `name` (a word); `country` (a word); and `events` (a *sentence* containing the names of all events the athelete participates in). Here is the constructor:

```
(define (make-athlete name country events)
  (append (list name country) (cons events '())))
```

(a) Write the appropriate selectors for the `athlete` ADT.

(b) We have created a list of `athletes` as follows:

```
(define all-athletes
       (list
         (make-athlete 'Picabo-Street 'USA (se 'slalom 'downhill))
         (make-athlete 'Todd 'USA (se 'slalom 'super-monkey-ball))
         (make-athlete 'Johnny-Moseley 'USA (se 'moguls 'video-games))
         ... more athletes ... ))
```

We wrote `get-participants` that takes an event and list of `athletes` as input and returns a *sentence* containing the names of all `athletes` who participate in the events. E.g.,

```
> (get-participants 'slalom all-athletes) ⟹ (Picabo-Street Todd)
```

Complete the definition of `get-participants`. **Respect the Data Abstraction!**

```
(define (get-participants event list-of-athletes)

   (cond ((____________________________ list-of-athletes) '())

         ((member? event (__________ (__________ list-of-athletes)))

          (________ (______________ (______________ list-of-athletes))

                    (get-participants event (______ list-of-athletes)))

         (else (get-participants event (__________ list-of-athletes)))) )
```

Question 5 — I can't talk right now, my head is in a vice... (8 points):

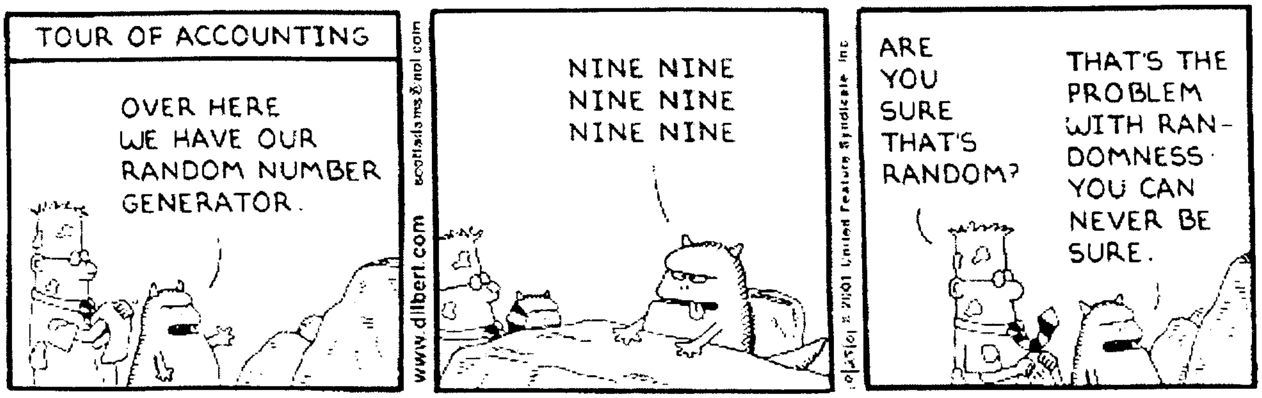
This question concerns the twenty-one project (the version without jokers).

The CS61A staff is going to Las Vegas, and we plan to play **a lot** of twenty-one! One of the questions in the project asked you to create your own strategy. We would like to figure out which of these strategies is the best, so we can use it in Vegas and win billions.

The procedure `best` takes an integer `n` and a non-null list of strategies as arguments. `best` will play `twenty-one` with each strategy `n` times and return the strategy that had the best results—in the case of a tie, any winner may be returned.

Fill in the blanks to complete the definition of `best`. Remember, you can assume that the list of strategies is not empty. **Do not create any new procedures!**

```
(define (best n list-of-strats)

  (define (helper best-strat best-result strats-left)

      (if (_______________________________________________)

          ________________________________________________

          (let ((result __________________________________))

             (if (< ____________________ ____________________)

                 _________________________________________

                 _________________________________________)) ))

  (helper ________________________________________________) )
```


TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
scottadams@aol.com
www.dilbert.com
NINE NINE NINE NINE NINE NINE
© 2001 United Feature Syndicate Inc
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS. YOU CAN NEVER BE SURE.

Question 6 — Dial `1-800-call-all` and save a buck or two... (8 points):

Louis Reasoner is trying to write a function `call-all` that takes a list of single-argument functions and returns a new unary function. The function returned by `call-all` should call each of the function in the list with its input and return a list of the results. E.g.,

```
> (define (identity x) x)
> (define (square   x) (* x x))
> (define (cube     x) (* x x x))
> (define powers (call-all (list identity square cube)))
> (powers 2)
(2 4 8)
> (define pigl-and-reverse (call-all (list pigl reverse)))
> (pigl-and-reverse 'class)
(assclay ssalc)
> (define no-results (call-all '()))
> (no-results 100)
()
```

Louis' first attempt at writing `call-all` looks like:

```
(define (call-all L)
  (if (null? L)
1       '()
2       (lambda (x)
3         (cons ((car L) x)
4               (call-all (cdr L)))) ))
```

(a) Complete the sentence below to describe what Louis' program returns versus what it is supposed to return (according to the specifications). If the expression produces an error, just say "error"; if it returns a procedure, just say "procedure"; if it causes an infinite loop, just say "loop". "A call to `((call-all (list square 1+)) 2)`

returns ______________________________

but *should* return ______________________________."

(b) Make **as few changes as necessary** to fix the procedure so that it works as advertised on all inputs. You will probably not need to change all the lines.

Change line ____ to ______________________________

Change line ____ to ______________________________

Change line ____ to ______________________________

Change line ____ to ______________________________