CS61A Midterm #1   February 15, 2006

Question 1 (6points):

What will Scheme print in response to the following expressions? If an expression produces an error message, you may just write "error"; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just write "procedure"; you don't have to show the form in which Scheme prints procedures.

```
(keep (lambda (x) (or (even? x) (< (count x) 3) ) )
        '(1 12 123) )
```

____

```
(se '(procedures are) (first 'class) )
```

____

```
(every   (* x x) '(4 5 6) )
```

____

```
(every first   (keep   even?   '(23 48 12 87 6) )
```

____

```
(word   (first '(wish you) )   (bf   '(were here) ) )
```

____

```
(cond   ('comfortable 'numb) (hey you) (else money) )
```

____

Question 2 (3 points):

```
(define (funky a b c)
   (if a b (* c c) ) )

> (funky (* 2 2) (* 3 3) (funky #f (* 4 4) (* 5 5) ) )
```

How many times is * invoked…

In applicative order? _______ In normal order? ________

In actual Scheme? _______

Question 3 (4 points):

Circle the procedures below (if any) that generate an iterative process. Don't circle the ones (if any) that generate a recursive process.

```
(define (magic-number? num)
   (if (< num 0)
       #f
       (if (= num 0)
           #t
           (magic-number? (- num 26) ) ) ) )

(define (magic-number? num)
   (if (< num 0)
       #f
       (if (= num 0)
           #t
           (or (magic-number? (- num 3) ) (magic-number? (- num 7) ) ) ) ) )
```

Question 4 (3 points):

```
(define (mystery n m)
   (cond ( (= n m) (+ n m) )
         ( (< n m) (mystery n (- m 1) ) )
         (else (mystery (- n 1) m) ) ) )
```

Which of the following is loop invariant of **mystery**, defined above, which takes two integers n and m as arguments?

______A. m+n _______B. n-m

______C. min(m, n) _______D. max(m, n)

Question 5 (3 points): Circle **T** for **true** of **F** for **false** for each of the following.

T **F** A $\Theta(N)$ algorithm always runs faster than a $\Theta(2N)$ algorithm for large enough values of N.

T **F** A $\Theta(N)$ algorithm always runs faster than a $\Theta(N^2)$ algorithm for large enough values of N.

T **F** A$\Theta(1)$ algorithm always runs faster than a $\Theta(N)$ algorithm for large enough values of N.

Question 6 (6 points):

Write the predicate **no-duplicates?** that takes a sentence as its argument, and returns #t if and only if no work appears more than once in the sentence. For example:

```
STK>  (no-duplicates?  '(and your bird can sing) )
#t
STK>  (no-duplicates?  '(the fool on the hill) )
#f
```

Question 7 (7 points):

Write **make-customized-every**, a function that takes a predicate **pred** as its argument and returns a procedure that behaves like **every**, except that it applies its function argument **fn** only to those words in the sentence argument **sent** for which the **pred** returns **#t**. Words for which **pred** returns **#f** are retained in the returned sentence unchanged. For example:

STK> (define num-every (make-customized-every number?))
STK> (num-every square ‘(a 2 b 3 c 4))
(a 4 b 9 c 16)

Question 8 (7 points):

Write a procedure **poly** that takes as its argument a sentence of one or more numbers, the coefficients of a polynomials, and returns a procedure that takes a single number as argument and returns the value of that polynomial with the given number as its argument.

For example, the polynomial f(x) = $x^3 + 2x^2 + 3x + 4$ would be defined and used this way:

```
STK>   (define f   (poly '(1 2 3 4) ) )
STK>   (f 1)
10                              ; f( 1 ) = 1^3 + 2*1^2 + 3*1 + 4 = 10
STK>   (f -1)
2                               ; f( -1 ) = (-1)^3 + 2*(-1)^2 + 3*(-1) + 4 = 2

STK>   (define g   (poly '(1 0 -4) ) )        ; g( x ) = x^2 - 4
STK>   (g 2)
0                               ; g( 2 ) = 2^2 – 4 = 0
```

Hint: Another way to write the polynomial $ax^3 + bx^2 + cx + d$ is

$$x * (ax^2 + bx + c) + d$$