CS 61A Midterm #1 - September 14, 1998

Your name

login: cs61a-

Discussion section number

TA's name

This exam is worth 20 points, or about 13% of your total course grade. The exam contains five 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 seven 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.

Question 1 (3 points):

What will Scheme print in response to the following expressions? If an expression produces an error message or runs forever without producing a result, you may just say ``error''; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just say ``procedure''; you don't have to show the form in which Scheme prints procedures. Assume that no global variables have been defined before entering

these expressions (other than the predefined Scheme primitives).

```
(3 + 5)

(butfirst (first (butfirst '(i want to tell you))))

((lambda (a b c) (b a c)) 3 + 5)

(let ((a +)
      (b (a 3 4)))
  (a b b))

(cond ((< 4 7) 5)
      ((> 4 2) 6)
      (else 3))

(se ('tell 'me 'why))
```

Question 2 (2 points):

Imagine that there is a **primitive** procedure called `counter`, with no arguments, that returns 1 the first time you call it, 2 the second time, and so on. (The multiplication procedure *, used below, is also primitive.)

Supposing that `counter` hasn't been called until now, what is the value of the expression

```
(* (counter) (counter))
```

under applicative order?

under normal order?

Your name login cs61a-

Question 3 (4 points):

Consider the function defined as follows:

$$\begin{aligned} &\lg(1) && = 0 \\ &= \lg(\lfloor n/2 \rfloor) + 1, && n > 1 \end{aligned}$$

In this problem you will write two Scheme procedures to compute this function. **Use the algorithm shown above;** do not use mathematical functions not shown here, such as `expt` or `sqrt`. You may use `(floor x)` to compute $\lfloor x \rfloor$.

(a) Write a procedure that computes this function using a recursive process.

(b) Write a procedure that computes this function using an iterative process.

Question 4 (6 points):

(a) Write a procedure named `constant-fn?` that takes two arguments: a function of one argument, and a sentence of numbers. It should return `#T` if the value returned by the function is always the same for all of the numbers in the argument sentence. For example:

```
> (constant-fn? sqrt '(2 3 4 5 6))
#F
```

> (constant-fn? (lambda (x) 4) '(5 3 88 2 100 7 8 9)) #T

> (constant-fn? (lambda (a) (< a 10)) '(22 23 24 25 26 27)) #T

> (constant-fn? (lambda (a) (< a 10)) '(5 7 9 11)) #F

Question 4 continues on the next page.

Your name login cs61a-

Question 4 continued:

(b) Write a procedure named `make-constant-checker` that takes a function as argument. It should return a function that takes a sentence of numbers as argument, returning `#T` if the value returned by the original function is always the same for all of the numbers in the argument sentence. For example:

```
> (define not-spanning-ten (make-constant-checker (lambda (a) (< a 10))))
NOT-SPANNING-TEN
```

> (not-spanning-ten '(22 23 24 25 26 27)) #T

> (not-spanning-ten '(5 7 9 11)) #F

Question 5 (5 points):

Suppose you have a sentence of numbers, and you want to find a polynomial function that fits them. The first step is to find the smallest possible degree for such a polynomial. (As you should know, the degree of a polynomial is the highest power of x in it.) One way to do that is the technique of *finite differences* :

If all the numbers are equal, then the polynomial is of degree zero (i.e., it's a constant).

If not, take the differences between adjacent numbers in the sentence, forming a new sentence with one fewer number:

`#=12`&=12

```
> (differences '(1 4 9 16 25 36 49))
(3 5 7 9 11 13)
```

because 4-1 = 3, 9-4 = 5, and so on. The degree of the polynomial for the original sentence of numbers is one greater than the degree of the polynomial for this new sentence. So we keep taking differences:

`#=12`&=12

```
> (differences '(3 5 7 9 11 13))
(2 2 2 2 2)
```

This new sentence has degree 0, since all the numbers are equal. So the original sentence has degree 2, because we had to take differences twice before reaching a sentence of all-equal numbers. (In fact, the original sentence in this example fits the polynomial $y = x^2$.)

(a) Use your `make-constant-checker` from question 4 to implement a function `all-equal?` that takes a sentence of numbers as its argument and returns `#T` if all of the numbers are equal, or `#F` otherwise.

This question continues on the next page.

Your name login cs61a-

Question 5 continued:

(b) Write the function `differences` that takes a non-empty sentence of numbers as its argument and returns the sentence of differences, as described on the previous page.

(c) Using parts (a) and (b), write the function `degree` that takes a sentence of numbers as its argument, and returns the smallest degree of a polynomial that fits those numbers. For example:

```
> (degree '(5 5 5 5 5 5))
0
```

> (degree '(1 4 9 16 25 36 49)) 2