Test

Please write your name here:

Before you begin, please take a moment to look over the entire test so that you can budget your time.

In this test the names "LISP", "Scheme", etc. refer to the languages implemented in the textbook's interpreters. The "real" languages will always be referred to by the names "real LISP", "real Scheme" etc.

Answers to essay questions will be judged on their completeness and clarity. However, please do not feel obligated to fill up all the empty space on the page. Conversely, do not feel limited by the space on a page, since you may continue an answer on the back of a page if you wish.

problem	subject	points	score
1	scoping	5	
2	Types, lambda, and Scheme	30	
3	functional programming	35	
4	design extension to Scheme	30	
Total		100	

This exam (only) counts for 20% of your final grade.

1. (5 points) Consider a variant of Scheme with dynamic scoping. Would the interpreter for this variant of Scheme need to make a closure for a `lambda` expression? Why or why not? (Please give a *short* answer.)

2. Consider the following Scheme function

```
(set cn
  (lambda (n)
    (lambda (f)
      (lambda (a)
        (if (= n 0)
            a
            (f (((cn (- n 1)) f) a)))))))
```

   (a) (10 points) What is the type of `cn`? (Give your answer in the notation discussed for ML.)

   (b) (5 points) Is `cn` polymorphic? Why or why not? (Please give a *short* answer.)

   (c) (5 points) What is the result, if any, of the following expression?

   ```
   (((cn 3) (lambda (x) (+ x 4))) 7)
   ```

   (d) (10 points) What is the result, if any, of the following expression?

   ```
   (((cn 4) (lambda (x) (cons (+ 3 (car x)) x))) '(7))
   ```

3. (10 points) Write an function `ip` in APL (without using `while`, `if`, recursion, or assignments that modify variables) that takes a an integer `n` and a vector `v` of integers and returns the sum of the product of `n` and each of the elements of `v`. For example `(ip 3 '())` returns 0 and `(ip 3 '(1 2 3))` returns $3 \times 1 + 3 \times 2 + 3 \times 3$ (that is, 18).

4. (10 points) Fill in the blanks in the following Scheme definition of `ip`. For this version the argument `v` is a list of integers.

```
(set ip
  (lambda (n v)
     (combine
       ____________________________________________
       ____________________________________________
       ____________________________________________
      )))

(set combine (lambda (f sum zero)
     (lambda (l) (if (null? l) zero
       (sum (f (car l)) ((combine f sum zero) (cdr l)))))))
```

5. (15 points) Write `ip` in LISP without using `set` or `while`. Define any auxiliary functions you may use. (Remember that LISP doesn't have `lambda`.)

6. In this problem you will use your experience with `lambda` in Scheme to add a new kind of expression to Scheme. The idea behind the expression `nu` is that it is like `lambda`, but a `nu` expression can define a recursive function. The syntax of `nu` is as follows:

```
⟨expression⟩ ::=  ( nu ⟨variable⟩ ⟨arglist⟩ ⟨expression⟩ )
    | ...
⟨arglist⟩ ::=  ( ⟨variable⟩* )
```

Your first problem is to define the evaluation rules for Scheme with `nu`, based on the following examples and analogies with `lambda`. As a first example, the following defines the factorial function in Scheme extended with `nu`.

```
(set factorial
  (nu fac (x)
    (if (= x 0) 1 (* n (fac (- n 1))))))
```

The following expression returns 3:

```
(+ 0 ((nu id (x) x) 3))
```

The following expression never terminates (i.e., it goes into an infinite loop).

```
(+ 1 ((nu oops (x) (oops x)) 0))
```

As a final example, the following expression returns 16 (i.e., 2^4).

```
((nu expt (x y)
    (if (= y 1)
        x
        (* x (expt x (- y 1)))))
 2 4)
```

(The questions for you are on the following page.)

(a) (15 points) The value of a `nu` expression of the form `(nu v (`x_1 ... x_n`) e)` can be represented by a closure of the form:

$$\langle\langle \texttt{(lambda (}x_1 \ \ldots \ x_n\texttt{) e)}, \rho \rangle\rangle.$$

Describe the environment one must put in such a closure so that when it is applied using the usual application rules for Scheme, it would work as desired (e.g., for the above examples). Illustrate your answer by writing the closure for the following expression.

```
(nu oops (x) (oops x))
```

(b) (15 points) Suppose we deleted `lambda` from Scheme and replaced it with `nu`. Describe the effects this would have on programming in Scheme.