Principles of Programming Languages (CS329) **Quiz 1**
Computer Science and Engineering **2003-08-29 11:30–13:00**
Indian Institute of Technology Bombay

This quiz has 4 pages. Write your answer clearly in the spaces provided. Do not attach rough work. Use the marks alongside each question for time management. Provide 1–2 sentences of informal justification to qualify for partial credit in case your final answer is wrong. You can use the FWH book and your own class notes only.

1. Draw the parse tree and the Stoy diagram of the λ-expression below, and on the parse tree, indicate each occurrence of an identifier as free or bound:

   (z (λ y ((λ z (x y)) z)))

	3

2. Reduce the following expression to normal form if possible, otherwise indicate that a normal form does not exist:

   (y (λ y (y (λ x (x y)))))

	1

3. Write a pure lambda expression to define the macro `M5?` which accepts a single Church numeral as argument and returns a boolean `TRUE` or `FALSE` according as the integer corresponding to the Church numeral is a multiple of five or not. Do not use recursion or the Y-combinator.

	2

4. What is the value of the following Scheme expression, assuming static scoping and applicative order evaluation?

```
(let ((x 5))
    (let ((x 6) (f (lambda (y z) (* y (- z x)))))
        (f x 8)))
```

	2

5. In class we have given a desugaring for the single `let` form `(let I=`E_a E_b`)`. Give a desugaring for the Scheme multiple `let` form

```
(let ((I1 E1)...(In En)) Eb)
```

where no `Ij` can be used in any `Ek` unless `Ij` is bound in an outer scope, in which case it takes its value from that outer binding.

	4

6. In class we saw how to desugar a single-parameter `(rec I` E`)` using the Y-combinator. We wish to extend this to a multi-parameter `letrec` form

```
(letrec ((I1 E1)...(In En)) Eb)
```

where each `Ek` is a single-argument `proc` and mutual recursion is freely permitted (i.e., any `Ij` can be used in any `Ek` and does **not** take its value from an outer binding). Give a syntactic translation from `letrec` into core FLK without `rec`. | | 4 |

7. In Scheme, a recursive Fibonacci function may be written as follows:

```
(define (fib n)
  (cond
    ((= n 0) 0)
    ((= n 1) 1)
    (else (+ (fib (- n 1)) (fib (- n 2)))) ) )
```

This is not tail-recursive, because the first `fib` call has to remember that the second `fib` and the addition are pending operations, and the second `fib` call must remember the pending addition. Convert the above code into continuation passing style by adding a continuation argument.

	4

Total: 20