Principles of Programming Languages (CS329) **Midterm exam**
Computer Science and Engineering **2004-09-15**
Indian Institute of Technology Bombay **14:30–16:30**

NAME _____ ROLL _____

This exam has 4 page/s. Write your answer clearly in the spaces provided and on the last blank page, if any. Do not attach rough work. Do not write inside the boxes meant for grading. 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 any textbook and your own class notes only, but not xeroxed copies of notes written by someone else, or printouts of material posted on the Web site.

1. We have seen how to represent non-negative integer n as a Church numeral $\bar{n}$. Unfortunately, $\bar{n}$ takes space linear in n to store. We can be smarter about storing $\bar{n}$ within $O(\text{poly}\log n)$ space, e.g.,

   $\bar{7}$ = `(λ f (λ a ( (2̄ (2̄ f)) ((2̄ f) (f a) ) ) ) )` or
   $\bar{8}$ = `(λ f (λ a ((2̄ (2̄ (2̄ f))) a)))`

   Assume applicative order reductions.

   (a) How many reduction steps are needed to evaluate `ZERO?` and `ODD?` as defined in class:

   ```
   ZERO? ≡ (λ n ((n (λ x FALSE)) TRUE))
    ODD? ≡ (λ n ((n NOT) FALSE) )
   ```

   Give your answer asymptotically wrt n. | | 1 |

   (b) Write down a lambda expression for $\hat{n}$ where $(\hat{n}\ \bar{2}) \equiv \bar{n}$. | | 1 |

   (c) Write down an implementation of `ZERO?` using $\hat{n}$ which needs only $O(s_n)$ reduction steps where s is the number of nodes in the AST of $\bar{n}$, and establish this performance bound. | | 1 |

(d) Similarly, write down an implementation of `ODD?` using $\hat{n}$ which needs only $O(s)$ reduction steps where s is the number of nodes in the AST of $\bar{n}$, and establish this performance bound. | | 1 |

2. We will design a dynamically scoped variant of FL which is slightly different from the "standard" one discussed in class.

(a) Under dynamic scope as defined in class, what would this expression evaluate to?

```
let f = (let a = 1 (lambda x (+ x a)) )
  (f 3)
```

| | 1 |

(b) Suppose we want the expression above to evaluate to 4, and the expression below to evaluate to 5.

```
let f = (let a = 1 (lambda x (+ x a)) )
  let a = 2
    (f 3)
```

Write a modified version of $\mathcal{E}$ ⟦`(lambda I` E`)`⟧ to achieve this. | | 1 |

3. Consider the following fragment of Java code.

```
class Sub { public int v; public Sub(int v_) { v=_v; } }
void f2(Sub s)   { s.v = 555; }
void f3(Sub s)   { s = new Sub(747); }
void f4(Sub a[]) { a[1].v = 737; }
void f5(Sub a[]) { a[2] = new Sub(777); }
void f6(Sub a[]) { a = new Sub[3]; a[0] = a[1] = a[2] = new Sub(10); }
// ... and in main ...
Sub b[] = new Sub[3];
for ( int i=0; i<3; i++ ) { b[i] = new Sub(i); }
```

Write down the output after the fragment above is executed, followed by statements (3a–3e) below, one after another with cumulative effect. `print` prints elements of an array in index order on a single line. Exact formatting is not required.

(a) `f2(b[0]); print(b);` | 1

(b) `f3(b[0]); print(b);` | 1

(c) `f4(b); print(b);` | 1

(d) `f5(b); print(b);` | 1

(e) `f6(b); print(b);` | 1

Answering this question should give you a good idea of Java's array storage and parameter-passing mechanisms.

4. In keeping with Java's behavior in Question 3, we will extend FL with Arrays and Indirect Reference, giving us the FLAIR language. The new syntactic constructs are as follows.

$$
\begin{array}{rcl}
E & \texttt{::=} & \ldots \\
& | & \texttt{(array } E_{\text{size}}\texttt{)} \\
& | & \texttt{(set! } E_{\text{array}}\texttt{[}E_{\text{index}}\texttt{]} \ E_{\text{val}}\texttt{)} \\
& | & E_{\text{array}}\texttt{[}E_{\text{index}}\texttt{]}
\end{array}
$$

(Caution: You may lose credit for inconsistent answers to the following questions.)

(a) Will $\mathcal{E}\ [\![\texttt{(lambda I } E\texttt{)}]\!]$ retain the pass-by-copy style discussed in class, or need to change? Justify. | 1

(b) Is there a need to recognize the syntactic form `(`E_p E_{array}`)`, i.e., passing a whole array into a function? Justify.

	2

(c) Likewise, is there a need to recognize the syntactic form `(`E_p E_{array}`[`E_{index}`])`, i.e., passing a single array element into a function? Justify.

	2

(d) Write $\mathcal{E}\ [\![\ldots]\!]$ rules for the new syntactic constructs above and any others that need to be modified. For convenience, assume we have a new-array Store accessor similar to new we have used earlier. Also design a suitable semantic type *Array* to which the syntax `(array...)` construct translates.

	4

Total: 20