Principles of Programming Languages (CS329) **Quiz 3**
Computer Science and Engineering **2003-11-14 11:30–12:45**
Indian Institute of Technology Bombay

NAME _____ ROLL _____

This quiz has 2 page/s. Write your answer clearly in the spaces provided and on any last blank page. 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. In the following code, an exception is looked up dynamically, and causes the termination of the innermost affected `try` block. What is the final value of the expression?

```
(try
  (let f = (proc x (+ 1000 (throw err x)))
    (try
      (call f 2)
     catch (err (z) (/ 500 z)))  );let
   catch (err (y) (/ 200 y)) ); try
```

	2

The answer is 250 (500/2). 1 mark for avoiding the 1000 (easy) and 1 mark for avoiding the wrong answer 100 (200/2).

2. What is the type of the following expression?

```
(lambda (g f) (lambda (x) (g (f x))))
```

Use the minimum possible number of free type variables. The type of a multi-parameter lambda `(lambda (x y)...)` is written as $T_x \times T_y \rightarrow \ldots$

	2

The answer is $(T_r \rightarrow T_s) \times (\boxed{\text{x}} \rightarrow T_r) \rightarrow \boxed{\text{x}} \rightarrow T_s$, with $\boxed{\text{f}} = \boxed{\text{x}} \rightarrow T_r$ and $\boxed{\text{g}} = T_r \rightarrow T_s$. 1 mark for a reasonable expression and 1 mark for minimum number of type variables with proper constraints between them.

3. We wish to design a language with exceptions latent or sleeping in variables. E.g., in this expression

```
let a = (exception 4)
  try
    let b = 2
      (+ a b)
  catch (x) (1+ x)
```

`a` packages an exception which set off whenever it is accessed, i.e., in `(+ a b)`, and returns `(1+ 4)` which is 5. There is only one kind of exception, so handlers need not mention names. But handlers are looked up dynamically as in FLaX. Our current language does not have any mutable store.

(a) Modify the signature of $\mathcal{E}$, which used to be

$$\mathcal{E} \quad : \quad \text{Expr} \to \mathit{SEnv} \to \mathit{Cont} \to \mathit{ExpVal}$$

by inserting an additional parameter to keep track of the current closest dynamic catcher. You should **not** need a general dynamic environment *DEnv* because there is only one kind of handler. For your reference, additional signatures for standard semantics without stores are given below:

$$\begin{aligned} \mathit{Cont} &= \mathit{ExpVal} \to \mathit{ExpVal} \\ \mathit{Proc} &= \mathit{ExpVal} \to \mathit{Cont} \to \mathit{ExpVal}. \end{aligned}$$

	1

$$\begin{aligned} h \in \mathit{Handler} &= \mathit{Cont} \\ \mathcal{E} \quad &: \quad \text{Expr} \to \mathit{SEnv} \to \mathit{Cont} \underline{\to \mathit{Handler}} \to \mathit{ExpVal} \end{aligned}$$

Reordering of parameters of $\mathcal{E}$ is ok. Setting $\mathit{Handler} = \mathit{Proc}$ is not ok if the call time continuation argument to handler *Proc*s are set wrongly (i.e., from inside `try` code). No partial credit.

(b) Write down new evaluation functions for these constructs: `I`, `(exception` E`)`, and `(try` E_t `catch (Ia)` E_h`)`.

	3

Define a new semantic domain $\mathit{Exception} \subset \mathit{ExpVal}$ with a constructor MakeException and a type test IsException.

$$\begin{aligned} \mathcal{E}\,[\![\texttt{(exception } E\texttt{)}]\!] &= \lambda u\, \lambda k\, \lambda h \left(\mathcal{E}\,[\![E]\!]\, u\, \underline{\lambda e\, (k\, (\textsc{MakeException}\ e))}\, h\right) \\ \mathcal{E}\,[\![\texttt{I}]\!] &= \lambda u\, \lambda k\, \lambda h\, \text{let } e = (u\ \texttt{I})\, \left(\text{IF } (\textsc{IsException}\ e)\ (h\ e)\ (k\ e)\right). \\ \mathcal{E}\,[\![\texttt{(try } E_t \texttt{ catch (Ia) } E_h\texttt{)}]\!] &= \lambda u\, \lambda k\, \lambda h \left(\mathcal{E}\,[\![E_t]\!]\, u\, k\, \underline{\lambda x \left(\mathcal{E}\,[\![E_h]\!] \begin{array}{|c|}\hline \texttt{Ia} \to x \\ \hline u \\ \hline \end{array} k\, \underline{h}\right)}\right) \end{aligned}$$

1 mark for each denotation; no partial credit. No marks for writing denotations of other syntactic forms, even if we have asked for them. We will be lenient about the last $\underline{h}$ (can be replaced by some junk). Variants of packaging exceptions will be accepted.

Total: 8