FOR INTERNAL SCRUTINY (date of this version: 4/12/2015)

UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR10061 ELEMENTS OF PROGRAMMING LANGUAGES

Tuesday 1 st April 2014

00:00 to 00:00

INSTRUCTIONS TO CANDIDATES

Answer QUESTION 1 and ONE other question.

Question 1 is COMPULSORY.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Year 3 Courses

Convener: ITO-Will-Determine
External Examiners: ITO-Will-Determine

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Using a BNF grammar, define the syntax of the untyped lambda-calculus (i.e. variables, lambda-abstraction, and application). *[3 marks]*

   (b) For each of the following pairs of expressions, indicate whether they are α-equivalent.

$$\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z \quad \equiv_\alpha^? \quad \texttt{if}\ x\ \texttt{then}\ z\ \texttt{else}\ y \qquad (1)$$

$$\lambda x.\lambda y.\lambda z.\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z \quad \equiv_\alpha^? \quad \lambda x.\lambda z.\lambda y.\texttt{if}\ x\ \texttt{then}\ z\ \texttt{else}\ y \qquad (2)$$

$$\lambda x.\lambda y.x + y \quad \equiv_\alpha^? \quad \lambda y.\ \lambda z.\ y + z \qquad (3)$$

$$\lambda x.\lambda y.x + y \quad \equiv_\alpha^? \quad \lambda x.\ \lambda x.\ x + y \qquad (4)$$

[4 marks]

   (c) Explain, in words, what is wrong with the following evaluation step, and how to correct the problem.

$$(\lambda x.\lambda y.x + y)\ (y + 1) \mapsto (\lambda y.(y + 1) + y)$$

[3 marks]

   (d) For each of the following three evaluation strategies, give a short definition and list one advantage of each approach. *[9 marks]*

      i. Call-by-value
      ii. Call-by-name
      iii. Call-by-need

   (e) Write the small-step operational semantics rules for call-by-name evaluation for the untyped lambda-calculus. *[6 marks]*

2. (a) Consider the following syntax for expressions involving arrays:

$$e ::= \cdots \mid \mathtt{array}(e_1, e_2) \mid e_1[e_2] \mid e_1[e_2] := e_3$$

The expression $\mathtt{array}(n, v)$ builds a new array of n elements initialized to value v. The expression $arr[i]$ dereferences array arr to get element i. Finally, the expression $arr[i] := v$ updates array arr to set element i to v, and returns a unit value $()$.

i. Assume the type of arrays of values of type τ is written $\mathtt{array}[\tau]$. Give appropriate typing rules for these constructs. *[6 marks]*

ii. Give two possible subtyping rules for arrays, one illustrating covariant subtyping and the other illustrating contravariant subtyping. *[4 marks]*

iii. Explain whether subtyping for arrays should be covariant, contravariant, both, or neither. *[4 marks]*

(b) Consider the following Scala code, which involves both exceptions and mutable (`var`) variables:

```
var x = 0
object MyException extends Throwable
try {
  try {
    x = x + 1
    throw MyException
    x = x + 10
  } catch {
    case e: NullPointerException => x = x + 100
  }
  finally {
    x = x + 1000
  }
} catch {
  case e: MyException => x = x + 10000
}
```

i. Explain, in words, what happens when the above code is executed. *[4 marks]*

ii. In Scala, `catch` blocks are written using pattern matching against the *run-time type* of the exception. What other features of Scala or Java could be used to implement this? *[2 marks]*

iii. What value does `x` have after the code is executed? *[5 marks]*

3. (a) Consider the following Scala code:

```
1 | val y = 0;
2 | class A(x: Int) {
3 |  val z = x + y
4 |  def f(x: String) = z
  | }
5 | new A(y).f("z")
```

For each line, list all of the identifiers on the line and indicate whether they are binding or bound occurrences. [*5 marks*]

(b) Give a complete typing derivation for the following judgment, or argue that the expression is not well-formed:

$$\vdash \Lambda A.\lambda x{:}\texttt{bool}.\lambda y{:}A.\lambda z{:}A.\texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ z : \forall A.\texttt{bool} \to A \to A \to A$$

[*9 marks*]

(c) In the C/C++/Java family of languages, the following `do...while` construct is provided:

```
do {
  stmt
} while (exp)
```

This will evaluate the statement `stmt` and then test the Boolean value of expression `exp`; if the value is true, execution continues by evaluating the `do...while` statement again, otherwise execution continues.

i. Give operational semantics rules for `do...while` statements (extending the large-step semantics for while-programs) [*6 marks*]

ii. Show how to express a single `do { stmt } while (exp)` statement in terms of `while` and `if ... then ... else`. [*5 marks*]