UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR11101 ADVANCES IN PROGRAMMING LANGUAGES

Friday 26th May 2017

14:30 to 16:30

INSTRUCTIONS TO CANDIDATES

Answer any TWO of the three questions. If more than two questions are answered, only QUESTION 1 and QUESTION 2 will be marked.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Year 4 Courses

Convener: I. Murray
External Examiners: A. Cohn, A. Donaldson, S. Kalvala

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. [*This question is worth a total of 25 marks.*]

(a) The following code is written in the *Rust* language and aims to construct and manipulate some vectors, eventually calculating two dot-product values. However, it contains some errors and does not compile correctly.

```
1  // Declare type of two-dimensional vectors
2  struct Vector2D { x: f64, y:f64 }
3
4  // Function to calculate dot product (scalar product) of two vectors
5  fn dot_product ( a: Vector2D, b: Vector2D ) -> f64
6      { a.x * b. x + a.y * b.y }
7
8  fn main() {
9      let p = Vector2D{ x: 1.2, y:3.4 };   // Set up first vector
10     let q = Vector2D{ x:-4.5, y:7.6 };   // Set up second vector
11     let r = Vector2D{ x: 0.0, y:5.3 };   // Set up third vector
12
13     p.x = p.x + q.x + r.x;               // Shift first vector horizontally
14
15     let a = dot_product(p,q);            // Calculate scalar product
16     let b = dot_product(q,r);            // Calculate scalar product
17
18     println!("Scalar products: p.q={}, q.r={}",a,b)
19 }
```

The Rust compiler reports exactly two errors, both in the body of the main() function. Both arise from distinctive features of the Rust language — neither would be considered as errors in the equivalent C code.

Find these two errors, and for each one:

- Give the number of the line on which it occurs;
- Explain the nature of the error and identify the Rust feature involved;
- Suggest modifications to the code that fix the problem and will be accepted by the compiler.

[*15 marks*]

(b) Rust has core language features that ensure memory safety, avoid memory leaks, and prevent data races.

(i) Identify one such memory safety property and explain briefly how Rust language features guarantee that property. [*3 marks*]

(ii) What is a "memory leak"? Explain briefly how Rust language features avoid memory leaks. [*3 marks*]

(iii) What is a "data race"? Explain briefly how Rust language features prevent data races. [*4 marks*]

2. [*This question is worth a total of 25 marks.*]

(a) System F extends the simply-typed lambda-calculus with explicit polymorphism: terms that take a type as a parameter. This language is expressive enough to define conventional algebraic datatypes from scratch. For example, if we assume predeclared types *Int* of integers and *Bool* of booleans, then we can define a type *Prod* of pairs of these.

$$Prod \stackrel{def}{=} \forall X.(Int \to Bool \to X) \to X$$

Consider another type, *OptInt*, for an "optional integer" with the following operations:

$$\begin{aligned} none &: OptInt \\ some &: Int \to OptInt \\ isNone &: OptInt \to Bool\,. \end{aligned}$$

The idea is that an *OptInt* value can be either *none* or *some*(n) for any *Int* value n, with *isNone* as a test to see which of these it is.

Write a definition in System F for the *OptInt* type, similar to that given for *Prod*, and definitions for each of the operations listed. You may assume the types *Int*, *Bool* and constants *true*, *false* : *Bool*. [*8 marks*]

(b) Recent versions of Java provide facilities for programming with *lambda expressions*, *higher-order functions*, and *closures*. For each of these three give a one-sentence explanation of what it is, and briefly suggest an example. [*9 marks*]

(c) Here is a Java class to tabulate the results of a numerical function.

```
1  import java.util.function.IntToDoubleFunction;
2
3  public class Tabulator {
4
5      private int lower, upper;
6
7      public Tabulator(int from, int to) { lower=from; upper=to; }
8
9      public void tabulate(IntToDoubleFunction f) {
10         for (int i=lower; i<=upper; i++)
11             System.out.println("f(" + i + ") = " + f.applyAsDouble(i));
12     }
13 }
```

The tabulate method is a higher-order function. Is it first-order, second-order, or third-order? Explain why. [*2 marks*]

QUESTION CONTINUES ON NEXT PAGE

(d) The following code attempts to use a Tabulator to calculate for all numbers from 1 to 10 first their squares and then all their powers from 1 to 5.

```
1  Tabulator t = new Tabulator(1,10);
2
3  IntToDoubleFunction square = x -> x*x;
4
5  t.tabulate(square);
6
7  int n=1;
8  IntToDoubleFunction powern = x -> java.lang.Math.pow(x,n);
9
10 for (int k=1; k<=5; k++) { n = k; t.tabulate(powern); }
```

The types in this code are all correct, as are the automatic conversions between **int** and **double** values. Nevertheless, a limitation in Java means that the compiler reports an error and is unable to compile the combination of lambda expressions and higher-order functions used here.

What has gone wrong? Explain which code is causing the problem, and what limitation in Java means the compiler returns an error. [*6 marks*]

3. [*This question is worth a total of 25 marks.*]

Programmers J. and K. are writing Java to implement a Counter class. Here is J's proposed code, which compiles and executes successfully.

```
1  public class Counter {
2
3      private int n=0;
4
5      public void up()    { n = n+1; }
6      public void reset() { n = 0; }
7      public int read()   { return n; }
8  }
```

K. complains that this class is not *thread safe* and may cause problems in concurrent code.

(a) Suppose we have a Counter c with current value n=5. Give an example of how calls to the methods of c from two concurrent threads could lead to incorrect results. Include information about when each call starts and finishes, when n changes, and its final value. Explain briefly why the outcome you describe is incorrect. [*10 marks*]

(b) K. recommends using **synchronized** methods. Explain what happens when a synchronized method of an object is invoked, compared with an unsynchronized method. [*3 marks*]

(c) J. is worried that using **synchronized** methods can cause "bottlenecks", where code runs much more slowly. Is J. right or wrong? Explain your answer. [*3 marks*]

(d) J. and K. discover that it is not always necessary to synchronize all methods in a class. To make Counter thread safe, which methods need to be **synchronized** and which can be left as they are? Explain your choices. [*9 marks*]