UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

ADVANCES IN PROGRAMMING LANGUAGES

Saturday 9 May 2009

09:30 to 11:30

Year 4 Courses

Convener: D K Arvind
External Examiners: A Frisch, J Gurd

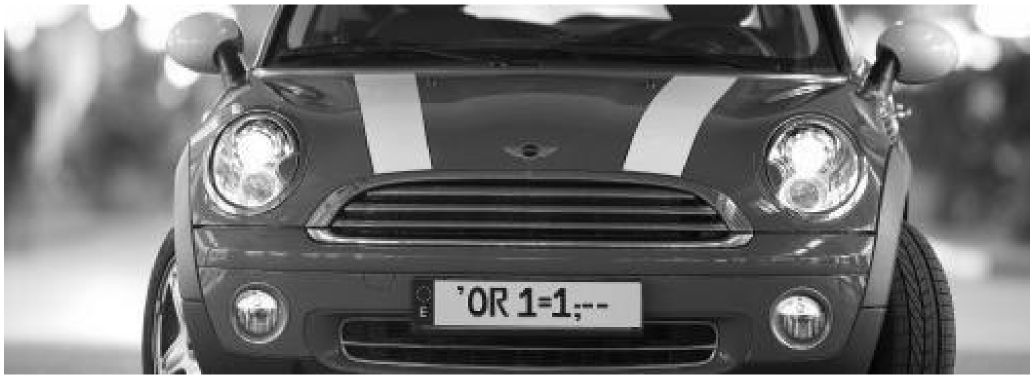
INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

1. (a) The following "photograph" appeared on various security blogs in 2008.



The supposed intention here is to carry out an *SQL injection attack* to disrupt the automatic scanning of car licence plates.

(i) What is SQL injection?

(ii) Suggest a possible malicious outcome of the above plate.

(iii) Give some sample code in Java or C# that would be susceptible to this attack. [*9 marks*]

(b) Write a brief description of the nature and uses of *metaprogramming*. In particular:

(i) What is metaprogramming?

(ii) Name two examples of metaprogramming; for each one, explain what it does, and state whether it acts at compile time or run time.

(iii) Describe how a language feature might help programmers make safer use of metaprogramming.

(iv) Explain (quasi)quotation and antiquotation, with an example in either LISP, MetaOCaml, F#, or other programming language of your choice. [*16 marks*]

2. (a) From the manual for the non-existent *SubGene* programming language:

> SubGene is strongly typed, with both subtyping and parameterized types. To handle the combination of these features, SubGene includes variance indicators in the form T(+X,−Y,Z). Here T is a type constructor, parameterized over X, Y and Z. The + on type parameter X indicates covariance, the − on Y shows contravariance, and their absence shows that T is invariant in Z.

Explain the following terms. For each, give a simple example.

(i) Subtype
(ii) Parameterized type
(iii) Covariant
(iv) Contravariant
(v) Invariant

[15 marks]

(b) Java, like many other object-oriented languages, makes heavy use of object *references*, where the values passed to and returned from methods are references that point to objects, not the entire objects themselves. Sometimes this leads to unexpected behaviour and runtime errors. The ESC/Java2 tool can detect these potential errors at compile time; especially if the programmer includes appropriate JML annotations.

Study the code below. Objects in the Point class represents points on a plane; if a and b are two such points then executing a.mash(b) will modify both by exchanging the x coordinate of one with the y coordinate of the other.

```
public class Point {
    int x, y;

    //@ modifies x;
    public void mash (Point other) {
      int t = x;
      x = other.y;  // First warning
      other.y = t;  // Second warning
}
```

When presented with this class, ESC/Java2 identifies two distinct problems, indicated by the warnings on the lines shown.

(i) What is the potential problem reported by the first warning?
(ii) Give a JML annotation that would avoid the first problem.
(iii) What is the potential problem reported by the second warning?
(iv) Give an example of code that would trigger the second problem.

[10 marks]

3. (a) Explain two motivations for concurrent programming that are *not* to do with utilising multiple processing units. [*4 marks*]

(b) Recall that a *future* is a concurrency abstraction for an asynchronously executing task, that at some point in the future will return a result. Futures are usually thread safe.

The Java interface for futures (somewhat modified) looks like this:

```
public interface Future<V> {
  /** Waits if necessary for the computation to complete, and then
   * retrieves its result.
   * @throws CancellationException if the task has been cancelled.
   */
  V get() throws CancellationException;

  /**
   * Attempts to cancel execution of this task. This attempt will fail if
   * the task has already completed, has already been cancelled, or could not
   * be cancelled for some other reason.
   * @return false if the task could not be cancelled, typically because it has
   * already completed normally; true otherwise  */
  boolean cancel();

  /** @return true if this task was cancelled before it completed */
  boolean isCancelled();

 /** @return true if this task completed or is cancelled */
  boolean isDone();
}
```

You are asked to implement an abstraction which combines two futures into a new one. Specifically, you are asked for a class FutureStringCombo which implements Future<String> and which has a constructor that takes two Future<String> futures as arguments. The resulting string should be the concatenation of the two results from the argument futures.

(i) Provide an implementation of FutureStringCombo. [*12 marks*]

(ii) Discuss whether you consider that your implementation is thread safe or not. If you consider that it is, justify carefully why. If you consider that it is not, explain what can go wrong. [*5 marks*]

(c) The future abstraction can be easily implemented in Polyphonic C#. Give a chord that contains the `get()` method and explain briefly how it works. (**Hint:** recall the one-place buffer example from lectures.) [*4 marks*]