UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

ADVANCES IN PROGRAMMING LANGUAGES

Thursday 15 May 2008

14:30 to 16:30

Year 4 Courses

Convener: D K Arvind
External Examiners: J Gurd, M Wooldridge

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

1. (a) Consider the following small Java program.

```
1     static void rotate(Object[] a) {   // Shift along all elements in the array
2       Object z = a[0];                 // Pick out the first element
3       for (int i=1; i<a.length; i++)   // Work through array
4       { a[i−1] = a[i]; }               // Move along everything else
5       a[a.length−1] = z;               // The first shall be last
6     }
7
8     static void clear(Object[] a) {    // Clear all elements of array
9       for (int i=0; i<a.length; i++)   // Work through the array
10      { a[i] = ""; }                   // Clear each element
11    }
12
13    static public void main(String[] args) {
14      String[] p = { "Hello", "World" };
15      Boolean[] q = { Boolean.FALSE, Boolean.FALSE, Boolean.TRUE };
16      rotate(p);
17      clear(p);
18      rotate(q);
19      clear(q);
20    }
```

This program compiles without any errors or warnings, but when executed it raises a type error concerning the use of arrays. What error is this? On which line does it occur?

For each of the four method calls in the main method, state whether or not it executes successfully, and why. [*8 marks*]

(b) For structured data Java uses *nominative typing*, while OCaml uses *structural typing*. Explain what these mean, with reference to the following type declarations:

```
class pair1 { int x; int y; }      // Pair of integers in Java
class pair2 { int x; int y; }

type pair1 = int * int             (* Pair of integers in OCaml *)
type pair2 = int * int
```

Write examples of code in each language, using these types, that illustrate the difference in behaviour. [*7 marks*]

(c) OCaml applies structural typing to objects through the use of *row polymorphism*. The following small language of types captures the essence of

this.

$$\begin{aligned}\tau &::= \alpha \quad | \quad \tau \times \tau \quad | \quad \tau \to \tau \\ &\quad | \quad \langle m_1 : \tau_1, \ldots, m_k : \tau_k \rangle \quad | \quad \langle m_1 : \tau_1, \ldots, m_k : \tau_k \mid \rho \rangle \\ \sigma &::= \forall \vec{\alpha} \vec{\rho} . \tau\end{aligned}$$

Here τ is a type and σ is a type scheme.

i. Consider the following field selection function and its type:

$$\lambda x.(x\#m) : \forall \alpha \forall \rho.\ \langle m{:}\alpha \mid \rho \rangle \to \alpha$$

What are α and ρ here? Explain the meaning of this type, and in particular what kind of arguments can be passed to the function. [*5 marks*]

ii. What is the action of the following function?

$$\lambda x.x\#\mathit{fun}(x\#\mathit{val})$$

What kind of arguments does it accept? Write down its type. [*5 marks*]

2. (a) The *Hoare triple* $\{P\}\ C\ \{Q\}$ has three constituent components: P, C and Q. State what each of these are, and explain the meaning of the triple itself. [*4 marks*]

Fill in appropriately the blanks in the following Hoare triples.

$$\{\quad\}\ \mathsf{a} := (\mathsf{b}+\mathsf{c})/2\ \{\mathsf{a} > 1\}$$
$$\{\quad\}\ \mathsf{u} := 2\mathsf{u};\ \mathsf{v} := \mathsf{v}+1;\ \{\mathsf{v} > \mathsf{u} \wedge \mathsf{u} > 0\}$$
$$\{\mathsf{a} = x \wedge \mathsf{b} = y \wedge y > 0\}\ \textbf{while}\ \mathsf{b}{>}0\ \textbf{do}\ (\mathsf{a} := 2{*}\mathsf{a};\ \mathsf{b} := \mathsf{b}{-}1)\ \{\quad\}$$

Make your specifications as strong as possible. You may assume that all variables take only integer values. [*5 marks*]

(b) Hoare triples are conventionally derived using axioms and rules such as

$$\frac{}{\{P\}\ \mathit{skip}\ \{P\}} \qquad \text{and} \qquad \frac{\{P\}\ C\ \{Q\} \qquad \{Q\}\ C'\ \{R\}}{\{P\}\ C; C'\ \{R\}}\ .$$

Explain the meaning of the following:

- A triple is *derivable* $\vdash \{P\}\ C\ \{Q\}$.
- A triple is *valid* $\vDash \{P\}\ C\ \{Q\}$.

Adding new constructions to a programming language requires extending the axioms and rules of Hoare logic. What does it mean for these rules to be *sound*? When are they *complete*? [*8 marks*]

(c) Here is a small JML specification for a function f taking a single integer argument arg.

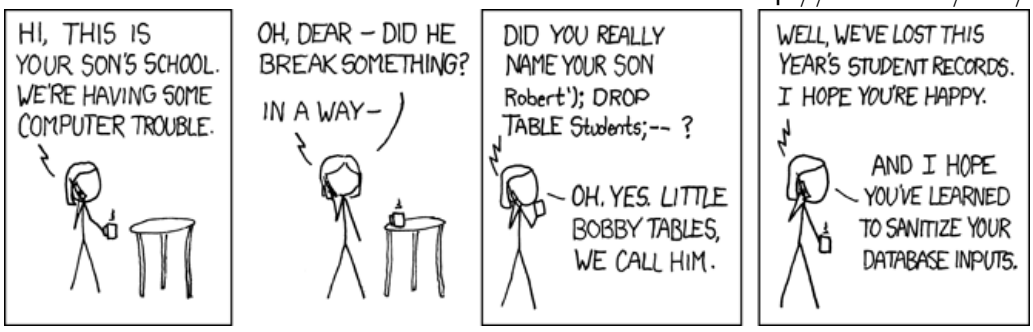
```
public int[] contents;

/*@ requires (\forall int i,j; 0<i && i<j && j<contents.length;
  @                              contents[i] <= contents[j]);
  @
  @ ensures contents[\result] == arg || \result == -1
  @*/
public int f (int arg) { ... }
```

Explain the meaning of the **requires** and **ensures** clauses in this specification. [*4 marks*]

Give a description in English of a function f which satisfies the specification. [*2 marks*]

How could a class in which this appears make use of a JML **invariant** specification? [*2 marks*]

3. (a) http://xkcd.com/327/



Explain the computer trouble that Bobby's school is having. Give an example of some code in Java or C# that might cause this sort of problem. [*6 marks*]

In the final frame, Mrs Roberts refers to "sanitizing" database inputs. What does this mean? Give one reason why sanitization is itself tricky to carry out correctly. [*4 marks*]

(b) The following C# code uses the LINQ language extensions to perform an SQL query.

```
Table<Student> students = con.GetTable<Student>()

var query = from s in students
            where s.Year = year && s.Course = courseTitle
            select new { s.Matric, s.Email };

foreach(var item in query)
{ Console.WriteLine("{0}: {1}", s.Matric, s.Email); }
```

How does this avoid the problem presented in your code for (a)?

Describe one additional advantage of this language-integrated approach to query management. [*5 marks*]

(c) The SQL-like query syntax above expands into the following C#:

```
var query = students.Where(s => (s.Year = year && s.Course = courseTitle))
                    .Select(s => new { s.Matric, s.Email });
```

This uses further language features new to C#, such as: lambda expressions, object initialization expressions; anonymous types; type inference.

Identify where in the code each of these occurs, and state briefly what they are. [*10 marks*]