UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

MEng Honours Degrees in Computing Part IV
MSc in Advanced Computing
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER C471

ADVANCED ISSUES IN OBJECT ORIENTED PROGRAMMING

Monday 10 May 2004, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1 A summary of the language L_2 as taught in the course appears on pages 5-7 of this paper.

a Consider the following program in L_2:

```
class A{
      A next; int i;
      A aux(int x){ this }
      A set(int x){ this.aux( this.i = x ) }
      A m(int x){ ( this.next = new A ).set(x) }    }
```

Assume that **`int`** has the usual meaning, and that the address ι_{10} is free. Write the contents of the store after evaluation of the following expression:

```
new A.set(1).m(2).m(3)
```

b Consider the classes `B` and `C`:

```
class B extds A{
      int j;
      A m(int x){ ( this.next = new B).set(x) }    }
class C ext B{ }
```

Write the contents of the store after evaluation of the following expression:

```
new C.set(1).m(2).m(3)
```

c Why was the above class B not defined as

```
class B extds A{
      int j;
      B m(int x){ ( this.next = new B).set(x) }    }
```

d Extend L_2 to express type casts as in Java. Type casts are expressions of the form `(TypeName)expr`; the type of that expression is `TypeName`. If execution of `expr` returns a value of type `TypeName` or a subtype, then execution of `(TypeName)expr` returns that value; otherwise it throws the error `castErr`.

For example, assuming that `x` was declared of type `A`, and assuming that semicolon (;) has the expected meaning:

Expression	**type**	**execution**
`x = new A; (A)x`	`A`	ok
`x = new A; (B)x`	`B`	`castErr`
`x = new B; (B)x`	`B`	ok
`x = new A; x.j`	type error	
`x = new B; x.j`	type error	
`x = new A; ((B)x).j`	`int`	`castErr`
`x = new B; ((B)x).j`	`int`	ok

The four parts carry, respectively, 15%, 10%, 15%, and 60% of the marks.

2a Consider the following C++ classes and declarations:

```
class A{ public:
    int fa; int faa;
    int f(){ … }
    virtual int g( ){ … }
    virtual int h( ){ … }
    virtual int k( ){ … }    };

class B: public A{ public:
    int fb;
    virtual int g( ){ … }
    virtual int m( ){ … }    };

class C: public A{ public:
    int fc;
    virtual int k( ){ … }
    virtual int n( ){ … }
    virtual int m( ){ … }    };

A a, *ap; B b, *bp; C c, *cp;
```

i) Sketch the layout (including virtual tables) of `B` objects, and `C` objects.

ii) For the following nine expressions say which are type incorrect (if any), and give the representation for those that are type correct:

```
ap=bp;  bp=ap;  a=c;
a.fa; c.fa; c.f(); cp->f(); c.h(); cp->k();
```

b Consider:

```
class D: public A, public B, public C { public:
    int fd;
    virtual int g( ){ … }    };

D d, *dp;
```

i) Sketch the layout (including virtual tables) of `D` objects.

ii) For the following ten expressions say which are ambiguous (if any), and give the representation for those that are unambiguous:

```
d.fa; d.fd;
ap=dp; cp=dp; bp=dp; a=d;
d.k(); dp->m(); dp->n(); dp->g();
```

c Consider:

```
class E: public A{ public: virtual void l(){…} };
class F public E, public B, public C { public:
    int ff;
    virtual int g( ){ … }    };

E e, *ep; F f, *fp;
```

i) Sketch the layout (including virtual tables) of `F` objects.

ii) For the following six expressions say which are ambiguous (if any), and give the representation for those that are unambiguous:

```
f.fa; f.ff; ap=fp; ep=fp; fp->l(); fp->g();
```

The three parts carry, respectively, 35%, 35% and 30% of the marks.

3 This question is about the Java bytecode and its verification as formalized in terms of $JVML_{00}$. The operational semantics and type system of $JVML_{00}$ is given on page 8 of this paper.

a Write the bytecode for method mb from the following Java classes:

```
class A{
    int fa;
    void ma(int i){ };    }

class B extends A{
    B fb;
    void mb( ){ ma( fb.fa ); }    }
```

b Consider the following $JVML_{00}$ code, where the receiver is of class A, the argument is of class B, and A and B are subclasses of C:

```
1  load 1
2  if 5
3  load 0
4  store 1
5  load 0
6  if 1
7  halt
```

Give a typing, if one exists; otherwise explain why the code is type incorrect.

c Consider heaps h, which map addresses to objects. Objects contain their class, and values for each pair of field and class identifier. Values are integers (i.e. **int** values or addresses):

$$\begin{aligned} \mathit{Heap} &= \mathit{Address} \;\texttt{->}\; \mathit{Object} \\ \mathit{Object} &= [[\;(\;\mathit{fldId}\;,\;\mathit{clssId}\;:\;\mathit{val}\;)*\;]]^{\mathit{clssId}} \end{aligned}$$

For example, $[[\;\texttt{fa},\;\texttt{A}\;:\;3\;]]^{\texttt{A}}$ is an object of class A, with 3 as the value of the field fa declared in class A.

i) Give the representation for an object of class B.

ii) Give the representation for an object of class C, where

```
class C extends B{ int fb; }
```

iii) Give operational semantics for the instructions `getfield C,type,f`, and `putfield C,type,f`. Remember: `getfield C,type,f` pops the top of stack and interprets it as an address, then pushes onto the stack the field with identifier f from class C from the object at that address. `putfield C,type,f` pops a value from the top of stack, then pops the top of the stack and interprets it as an address, then overwrites in the object at that address the field f from class C with the value.

iv) Give type rules for `getfield C,type,f`, and `putfield C,type,f`.

The three parts carry, respectively, 15%, 20%, and 65% of the marks.

4 This question is about the Java dynamic linking phases.

a i) What subtype checks does the verifier perform?
  ii) What subtype checks does the verifier *not* perform?
  iii) What checks does resolution perform?

b Consider the following Java classes:

```
class A1{ }                    class A2 extends A1{   }
class A3 extends A2{ }         class A4 extends A3{   }

class B1{ }                    class B2 extends B1{   }

class C1{
   A1 f;
   A1 m1(){  A1 a=new A4(); return new A2(); }   }

class C2 extends C1{
   void m2( ){ m3(new B2()); }
   void m3(B1 b){    }   }

class D{
   void m1(){
       System.out.println(" D.m1-a");
       new C1().f = new A3();
       System.out.println(" D.m1-b"); }
   void m2(){ System.out.println(" D.m2 "); }   }

class Test{
    public static void main(String[] args){
        System.out.println("- 1");  ff(false, null);
        System.out.println("- 2");  ff(false, new C2());
        System.out.println("- 3");  ff(true, new C2());
        System.out.println("- 4");  new D().m1();
        System.out.println("- 5");  }

    static void ff(boolean x, C2 c){
        if (x){ System.out.println(" Test.ff-a");
                new D().m2( );
                System.out.println(" Test.ff-b");} }   }
```

Write out the outcome of verbose execution of the method `main` from `Test`.

c Consider that after compiling all the classes in part b, the following, modified version of class `C1` is compiled:

```
class C1{ A1 m1(){  A1 a = new A4(); return new A2(); }   }
```

Write out the outcome of verbose execution of the method `main` from `Test`.

d Why does the verifier not perform the checks performed by resolution (i.e. those checks you mentioned in part a(ii)?

The four parts carry, respectively, 10%, 40%, 30% and 20% of the marks.

The Syntax of $\mathcal{L}_2$

progr	::=	*class**
class	::=	class *c* extds *c* { *field** *meth** }
field	::=	*type f*
meth	::=	*type m* (*type* x) { *e* }
type	::=	bool | *c*
e	::=	if *e* then *e* else *e* | *var* := *e* | *e* .*m* (*e*)
	|	new *c* | *var* | this | true | false | null .
var	::=	x | *e* .*f*

Field and method lookup functions

For program P, identifiers c, f, m and $\mathcal{C}(\mathsf{P},\mathsf{c}) = \text{class c extds c}'$ we define:

$$
\begin{array}{lcl}
\mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) & = & \begin{cases} \mathsf{t} & \textit{if } \text{cBody} = \ldots \mathsf{t\ f} \ldots \\ \mathcal{U}df & \textit{otherwise} \end{cases} \\
\mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) & = & \begin{cases} \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) & \textit{if } \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) \neq \mathcal{U}df, \\ \mathcal{F}(\mathsf{P},\mathsf{c}',\mathsf{f}) & \textit{otherwise} \end{cases} \\
\mathcal{F}(\mathsf{P},\mathsf{Object},\mathsf{f}) & = & \mathcal{U}df \\
\mathcal{F}s(\mathsf{P},\mathsf{c}) & = & \{\mathsf{f} \mid \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) \neq \mathcal{U}df\} \\
\mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) & = & \begin{cases} \mathsf{t\ m(\,t_1\ x)}\ \phi\{\ \mathsf{e}\ \} & \textit{if } \text{cBody} = \ldots \mathsf{t\ m\,(\,t_1\ x)\,\{\ e\ \}} \ldots \\ \mathcal{U}df & \text{otherwise} \end{cases} \\
\mathcal{M}(\mathsf{P},\mathsf{c},\mathsf{m}) & = & \begin{cases} \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) & \textit{if } \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) \neq \mathcal{U}df, \\ \mathcal{M}(\mathsf{P},\mathsf{c}',\mathsf{m}) & \textit{otherwise} \end{cases} \\
\mathcal{M}(\mathsf{P},\mathsf{Object},\mathsf{m}) & = & \mathcal{U}df
\end{array}
$$

Values agreeing to a type

$$
\frac{}{\mathsf{P},\sigma \vdash \mathsf{true} \lhd \mathsf{bool}}
\qquad
\frac{}{\mathsf{P},\sigma \vdash \mathsf{false} \lhd \mathsf{bool}}
\qquad
\frac{\mathsf{P} \vdash \mathsf{t} \diamond_c}{\mathsf{P},\sigma \vdash \mathsf{null} \lhd \mathsf{t}}
$$

$$
\frac{\begin{array}{l} \sigma(\iota) = [[\ \ldots\ ]]^{\mathsf{c}} \\ \mathsf{P} \vdash \mathsf{c} \leq \mathsf{c}' \\ \forall \mathsf{f} \in \mathcal{F}s(\mathsf{P},\mathsf{c}) :\ \mathsf{P},\sigma \vdash \sigma(\iota)(\mathsf{f}) \lhd \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) \end{array}}{\mathsf{P},\sigma \vdash \iota \lhd \mathsf{c}'}
$$

The Operational Semantics of $\mathcal{L}_2$

val

$$\frac{}{\mathsf{v},\sigma \leadsto_{\mathsf{P}} \mathsf{v},\sigma}$$

cond$_1$

$$\frac{\begin{array}{l}\mathsf{e},\sigma \leadsto_{\mathsf{P}} \mathsf{true},\sigma'' \\ \mathsf{e}_1,\sigma'' \leadsto_{\mathsf{P}} \mathsf{v},\sigma'\end{array}}{\mathsf{if\ e\ then\ e_1\ else\ e_2},\sigma \leadsto_{\mathsf{P}} \mathsf{v},\sigma'}$$

cond$_2$

$$\frac{\begin{array}{l}\mathsf{e},\sigma \leadsto_{\mathsf{P}} \mathsf{false},\sigma'' \\ \mathsf{e}_2,\sigma'' \leadsto_{\mathsf{P}} \mathsf{v},\sigma'\end{array}}{\mathsf{if\ e\ then\ e_1\ else\ e_2},\sigma \leadsto_{\mathsf{P}} \mathsf{v},\sigma'}$$

var

$$\frac{}{\begin{array}{l}\mathsf{x},\sigma \leadsto_{\mathsf{P}} \sigma(\mathsf{x}),\sigma \\ \mathsf{this},\sigma \leadsto_{\mathsf{P}} \sigma(\mathsf{this}),\sigma\end{array}}$$

fld

$$\frac{\mathsf{e},\sigma \leadsto_{\mathsf{P}} \iota,\sigma'}{\mathsf{e.f},\sigma \leadsto_{\mathsf{P}} \sigma'(\iota)(\mathsf{f}),\sigma'}$$

ass

$$\frac{\mathsf{e},\sigma \leadsto_{\mathsf{P}} \mathsf{v},\sigma'}{\mathsf{x{:=}e},\sigma \leadsto_{\mathsf{P}} \mathsf{v},\sigma'[\mathsf{x}\mapsto\mathsf{v}]}$$

fldAss

$$\frac{\begin{array}{l}\mathsf{e},\sigma \leadsto_{\mathsf{P}} \iota,\sigma'' \\ \mathsf{e}',\sigma'' \leadsto_{\mathsf{P}} \mathsf{v},\sigma''' \\ \sigma'''(\iota)(\mathsf{f}) \neq \mathcal{U}df \\ \sigma' = \sigma'''[\iota\mapsto\sigma'''(\iota)[\mathsf{f}\mapsto\mathsf{v}]]\end{array}}{\mathsf{e.f{:=}e}',\sigma \leadsto_{\mathsf{P}} \mathsf{v},\sigma'}$$

new

$$\frac{\begin{array}{l}\mathcal{F}s(\mathsf{P},\mathsf{c}) = \{\mathsf{f}_1,...,\mathsf{f}_r\} \\ \forall \mathsf{l}\in 1,...,\mathsf{r}: \quad \mathsf{v}_l \text{ initial for } \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}_\mathsf{l}) \\ \iota \text{ is new in } \sigma\end{array}}{\mathsf{new\ c},\sigma \leadsto_{\mathsf{P}} \iota,\sigma[\iota\mapsto [[\ \mathsf{f}_1 : \mathsf{v}_1,...,\mathsf{f}_r : \mathsf{v}_r\ ]]^{\mathsf{c}}]}$$

null

$$\frac{\mathsf{e},\sigma \leadsto_{\mathsf{P}} \mathsf{null},\sigma'}{\begin{array}{l}\mathsf{e.f{:=}e}',\sigma \leadsto_{\mathsf{P}} \mathsf{nullPntrExc},\sigma' \\ \mathsf{e.f},\sigma \leadsto_{\mathsf{P}} \mathsf{nullPntrExc},\sigma' \\ \mathsf{e.m(e_1)},\sigma \leadsto_{\mathsf{P}} \mathsf{nullPntrExc},\sigma'\end{array}}$$

methCall

$$\frac{\begin{array}{l}\mathsf{e}_0,\sigma \leadsto_{\mathsf{P}} \iota,\sigma_0 \\ \mathsf{e}_1,\sigma_0 \leadsto_{\mathsf{P}} \mathsf{v}_1,\sigma_1 \\ \sigma_1(\iota) = [[\ ...\ ]]^{\mathsf{c}} \\ \mathcal{M}(\mathsf{P},\mathsf{c},\mathsf{m}) = \mathsf{t\ m(\ t_1\ x)\ \ \{\ e\ \}} \\ \sigma' = \sigma_1[\mathsf{this}\mapsto\iota][\mathsf{x}\mapsto\mathsf{v}_1] \\ \mathsf{e},\sigma' \leadsto_{\mathsf{P}} \mathsf{v},\sigma''\end{array}}{\mathsf{e_0.m(e_1)},\sigma \leadsto_{\mathsf{P}} \mathsf{v},\sigma''[\mathsf{this}\mapsto\sigma(\mathsf{this}),\mathsf{x}\mapsto\sigma(\mathsf{x})]}$$

The Type System of $\mathcal{L}_2$

litVarThis

$$\frac{P \vdash \Gamma \diamond}{\begin{array}{l} P,\Gamma \vdash \mathsf{true} : \mathsf{bool} \\ P,\Gamma \vdash \mathsf{false} : \mathsf{bool} \\ P,\Gamma \vdash x : \Gamma(x) \\ P,\Gamma \vdash \mathsf{this} : \Gamma(\mathsf{this}) \end{array}}$$

newNull

$$\frac{\begin{array}{l} P \vdash \Gamma \diamond \\ P \vdash c \diamond_c \end{array}}{\begin{array}{l} P,\Gamma \vdash \mathsf{null} : c \\ P,\Gamma \vdash \mathsf{new}\ c : c \end{array}}$$

fld

$$\frac{\begin{array}{l} P,\Gamma \vdash e : c \\ \mathcal{F}(P,c,f) = t \end{array}}{P,\Gamma \vdash e.f : t}$$

ass

$$\frac{\begin{array}{l} P,\Gamma \vdash x : t \\ P,\Gamma \vdash e : t' \\ P \vdash t' \leq t \end{array}}{P,\Gamma \vdash x{:=}e : t'}$$

fldAss

$$\frac{\begin{array}{l} P,\Gamma \vdash e : c \\ P,\Gamma \vdash e' : t' \\ \mathcal{F}(P,c,f) = t \\ P \vdash t' \leq t \end{array}}{P,\Gamma \vdash e.f{:=}e' : t'}$$

cond

$$\frac{\begin{array}{l} P,\Gamma \vdash e : \mathsf{bool} \\ P,\Gamma \vdash e_1 : t_1 \\ P,\Gamma \vdash e_2 : t_2 \\ P \vdash t_i \leq t \text{ for } i \in 1,2 \end{array}}{P,\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : t}$$

methCall

$$\frac{\begin{array}{l} P,\Gamma \vdash e_0 : c \\ P,\Gamma \vdash e_1 : t_1' \\ \mathcal{M}(P,c,m) = t\ m(\,t_1\ x)\ \{\ e\,\} \\ P \vdash t_1' \leq t_1 \end{array}}{P,\Gamma \vdash e_0.m(\,e_1) : t}$$

$\mathcal{JVML}_{00}$ **Syntax**

$$\textit{instruction} \; ::= \; \mathsf{inc} \mid \mathsf{pop} \mid \mathsf{store}\ x \mid \mathsf{load}\ x \mid \mathsf{if}\ L \mid \mathsf{halt}$$

$\mathcal{JVML}_{00}$ **Operational Semantics**

$$\frac{\mathsf{P}[pc] = \mathsf{inc}}{\mathsf{P} \vdash pc, f, n{\cdot}s \leadsto pc{+}1, f, (n{+}1){\cdot}s} \qquad \frac{\mathsf{P}[pc] = \mathsf{pop}}{\mathsf{P} \vdash pc, f, v{\cdot}s \leadsto pc{+}1, f, s}$$

$$\frac{\mathsf{P}[pc] = \mathsf{load}\ x}{\mathsf{P} \vdash pc, f, s \leadsto pc{+}1, f, f(x){\cdot}s} \qquad \frac{\mathsf{P}[pc] = \mathsf{store}\ x}{\mathsf{P} \vdash pc, f, v{\cdot}s \leadsto pc{+}1, f[x \mapsto v], s}$$

$$\frac{\mathsf{P}[pc] = \mathsf{if}\ L}{\mathsf{P} \vdash pc, f, O{\cdot}s \leadsto pc{+}1, f, s} \qquad \frac{\mathsf{P}[pc] = \mathsf{if}\ L, \;\; n \neq 0}{\mathsf{P} \vdash pc, f, n{\cdot}s \leadsto L, f, s}$$

$\mathcal{JVML}_{00}$ **Type System**

$$\frac{\begin{array}{c} \mathsf{P}[i] = \mathsf{inc} \\ \mathsf{P} \vdash F_i \leq F_{i+1} \\ \mathsf{P} \vdash S_i \leq S_{i+1}, \;\; \exists S:\ S_i = \mathsf{int}{\cdot}S \\ i+1 \in \mathit{Dom}(\mathsf{P}) \end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}} \qquad \frac{\begin{array}{c} \mathsf{P}[i] = \mathsf{if}\ L \\ \mathsf{P} \vdash F_i \leq F_{i+1}, \;\; \mathsf{P} \vdash F_i \leq F_L \\ \mathsf{P} \vdash S_i \leq \mathsf{t}{\cdot}S_{i+1}, \;\; \mathsf{P} \vdash S_i \leq \mathsf{t}{\cdot}S_L \\ i+1 \in \mathit{Dom}(\mathsf{P}),\ L \in \mathit{Dom}(\mathsf{P}) \end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

$$\frac{\begin{array}{c} \mathsf{P}[i] = \mathsf{pop} \\ \mathsf{P} \vdash F_i \leq F_{i+1} \\ \mathsf{P} \vdash S_i \leq \mathsf{t}{\cdot}S_{i+1} \\ i+1 \in \mathit{Dom}(\mathsf{P}) \end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}} \qquad \frac{\begin{array}{c} \mathsf{P}[i] = \mathsf{load}\ x \\ x \in \mathit{Dom}(F_i) \\ \mathsf{P} \vdash F_i \leq F_{i+1} \\ \mathsf{P} \vdash F_i[x]{\cdot}S_i \leq S_{i+1} \\ i+1 \in \mathit{Dom}(\mathsf{P}) \end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

$$\frac{\begin{array}{c} \mathsf{P}[i] = \mathsf{store}\ x \\ x \in \mathit{Dom}(F_i) \\ \mathsf{P} \vdash F_i[x \mapsto \mathsf{t}] \leq F_{i+1} \\ \mathsf{P} \vdash S_i \leq \mathsf{t}{\cdot}S_{i+1} \\ i+1 \in \mathit{Dom}(\mathsf{P}) \end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}} \qquad \frac{\mathsf{P}[i] = \mathsf{halt}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$