UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2003

MSci Honours Degree in Mathematics and Computer Science Part IV
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
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER C471

ADVANCED ISSUES IN OBJECT ORIENTED PROGRAMMING

Friday 9 May 2003, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1 The language L2 is outlined on pages 5-7 of this paper.

a Consider the following program `P` in L2:

```
class A{ A nxt; A m(A x){ this.nxt := new A } }
class B extends A{ A m(A x){ this.nxt := x}   }
```

Assume an initial store where **this** is mapped to an object of class `B`, and assume that `;` has the obvious meaning. Write the contents of the store after evaluation of the following expressions:

```
this.nxt := new A; x := new B; x.m(this.nxt);
```

b Consider a simplified version of Java arrays, whereby:

1) Arrays are one dimensional, and have length 10, i.e. from `0` to `9`.

2) Array types have the form `c[]`, where `c` is a class name.

3) Reading (resp. assigning) the `j`-th element of an array `e` has the form `e[j]` (resp. `e[j]:=...`) and requires `j` to be between `0` and `9`; otherwise an exception is thrown.

4) The expression **new** `c[]` creates an array of `10` objects of class `c`.

5) The type `c[]` is a subtype of `c'[]`, if `c'` is a subclass of `c`.

6) Assigning to the `j`-th element of an array of runtime type `c[]` replaces the `j`-th element, if the right hand side of the assignment is of a subtype of `c`, and otherwise throws an exception.

For example, assuming the obvious meaning for `;` and for local variables, the following terms will behave as indicated in the comments:

```
A[] as; A anA; B[] bs; B aB;
As := new A[]; bs := new B[];
a := new A; b := new B;
as[2] := a;  // OK
as[3] := b;  // OK
bs[4] := a;  // Type Error
bs := as;    // Type Error
as := new B[];
a := new B;  // OK
as[3] := a;  // OK
a := new A;  // OK
as[3] := a;  // Type correct, runtime exception
```

Extend the description of L2, so that is describes the above

i) Extend the syntax.

ii) Extend definition of store and operational semantics (assume comparison operations for `integer` values).

iii) Extend the type system (assume a predefined type `integer`).

iv) Extend the definition of a value agreeing with a type.

c Justify why the Java language requires the run-time check described in part b.6.

The three parts carry, respectively, 10%, 80% and 10% of the marks.

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

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

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

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

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

i) Sketch the layout of `B` objects, and that of `C` objects.

ii) For the following expressions say which are type incorrect, and give the representation for those that are type correct:
`ap=bp;  bp=ap;  a=c;`
`a.fa; c.fa; c.ff(); cp->ff(); c.f(); cp->f();`

b Consider:

```
class D { public:
    int fd;
    virtual int g( ){ … }       };

class E: public B, public C, public virtual D {
public:
    int fe;
    virtual int m( ){ … }     };

D d, *dp; E e, *ep;
```

i) Sketch the layout of `E` objects.

ii) For the following expressions say which are ambiguous, and give the representation for those that are unambiguous:
`ap=ep;  bp=ep;  e.fa; e.fd;`

c Consider:

```
class F: public E, public virtual D { public:
    int ff;
    virtual int q( ){ … }       };
```

and sketch the layout of `F` objects.

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

a Write the bytecode for method m from the following java classes:

```
class A{
    int fa;
}

class B{
    int fb;
    void m(B anA){
        anA.fa = anA.fa + fb;
        m(anA); }
}
```

b Give a typing for 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 0
2  load 1
3  if 6
4  pop
5  load 1
6  if 1
7  halt
```

c Why is any $JVML_{00}$ code containing the following pattern type incorrect (Hint: only consider type rules):

```
…    …
k  if 1
…    …
1  if k
…    …
```

d Assume an instruction `swapIf k` which swaps the two top stack elements, and then, without popping anything from the stack, if the new top of stack is different from 0, it continues at label k, and otherwise it continues as normal.

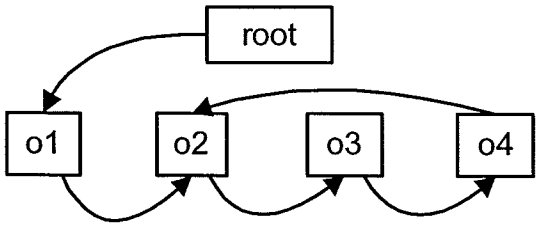
Give operational semantics and type rules for instruction `swapIf k`.

4a i) When does an object o *dominate* another object o′?

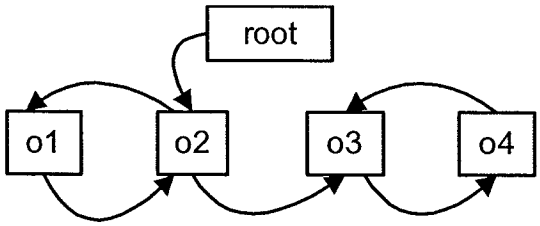
ii) What guarantees (in terms of domination) are given when object o *owns* another object o′?

iii) Given ownership relationships *own1*, and *own2*, such that o *own1* o′ implies that o *own2* o′. Which relationship gives more information (justify in one or two sentences)?

b i) Write out the domination relationship for the following object graph:

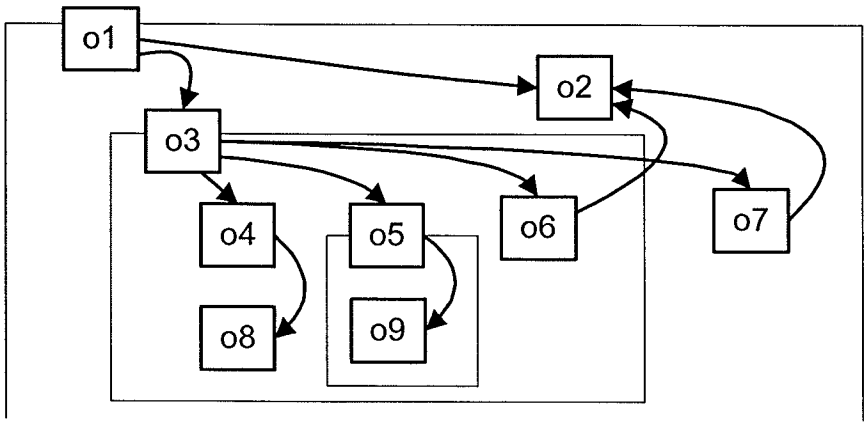


ii) Write out the domination relationship for the following object graph:



iii) Write out an ownership relationship which applies to the graphs from parts *i* and *ii*.

c Consider the following object graph, where the boxes indicate ownership:



i) Explain why an arrow from o2 to o6 would be illegal.

ii) Assume that objects o4, o6 and o7 belong to a class B, and o2, o8 and o9 belong to a class C, and o5 belongs to a class BB, and o3 belongs to a class A. Write class definitions for B, BB, and A.

The three parts carry, respectively, 30%, 35% and 35% 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:

$$\mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) = \begin{cases} \mathsf{t} & \textit{if } \mathsf{cBody} = \ldots\ \mathsf{t}\ \mathsf{f}\ \ldots \\ \mathcal{U}df & \textit{otherwise} \end{cases}$$

$$\mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) = \begin{cases} \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) & \textit{if } \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) \neq \mathcal{U}df, \\ \mathcal{F}(\mathsf{P},\mathsf{c}',\mathsf{f}) & \textit{otherwise} \end{cases}$$

$$\mathcal{F}(\mathsf{P},\mathsf{Object},\mathsf{f}) = \mathcal{U}df$$

$$\mathcal{F}s(\mathsf{P},\mathsf{c}) = \{\mathsf{f} \mid \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) \neq \mathcal{U}df\}$$

$$\mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) = \begin{cases} \mathsf{t}\ \mathsf{m}(\ \mathsf{t}_1\ \mathsf{x})\ \phi\{\ \mathsf{e}\ \} & \textit{if } \mathsf{cBody} = \ldots \mathsf{t}\ \mathsf{m}(\ \mathsf{t}_1\ \mathsf{x})\{\ \mathsf{e}\ \}\ldots \\ \mathcal{U}df & \text{otherwise} \end{cases}$$

$$\mathcal{M}(\mathsf{P},\mathsf{c},\mathsf{m}) = \begin{cases} \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) & \textit{if } \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) \neq \mathcal{U}df, \\ \mathcal{M}(\mathsf{P},\mathsf{c}',\mathsf{m}) & \textit{otherwise} \end{cases}$$

$$\mathcal{M}(\mathsf{P},\mathsf{Object},\mathsf{m}) = \mathcal{U}df$$

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

$$\frac{}{v,\sigma \rightsquigarrow_P v,\sigma}$$

cond_1

$$\frac{e,\sigma \rightsquigarrow_P \text{true},\sigma'' \quad e_1,\sigma'' \rightsquigarrow_P v,\sigma'}{\text{if } e \text{ then } e_1 \text{ else } e_2,\sigma \rightsquigarrow_P v,\sigma'}$$

cond_2

$$\frac{e,\sigma \rightsquigarrow_P \text{false},\sigma'' \quad e_2,\sigma'' \rightsquigarrow_P v,\sigma'}{\text{if } e \text{ then } e_1 \text{ else } e_2,\sigma \rightsquigarrow_P v,\sigma'}$$

var

$$\frac{}{x,\sigma \rightsquigarrow_P \sigma(x),\sigma}$$

$$\text{this},\sigma \rightsquigarrow_P \sigma(\text{this}),\sigma$$

fld

$$\frac{e,\sigma \rightsquigarrow_P \iota,\sigma'}{e.f,\sigma \rightsquigarrow_P \sigma'(\iota)(f),\sigma'}$$

ass

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

fldAss

$$\frac{e,\sigma \rightsquigarrow_P \iota,\sigma'' \quad e',\sigma'' \rightsquigarrow_P v,\sigma''' \quad \sigma'''(\iota)(f) \neq \mathit{Udf} \quad \sigma' = \sigma'''[\iota\mapsto\sigma'''(\iota)[f\mapsto v]]}{e.f{:=}e',\sigma \rightsquigarrow_P v,\sigma'}$$

new

$$\frac{\mathcal{F}s(P,c) = \{f_1,\ldots,f_r\} \quad \forall l \in 1,\ldots,r: \quad v_l \text{ initial for } \mathcal{F}(P,c,f_l) \quad \iota \text{ is new in } \sigma}{\text{new } c,\sigma \rightsquigarrow_P \iota,\sigma[\iota\mapsto [\![f_1 : v_1,\ldots,f_r : v_r]\!]^c]}$$

null

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

methCall

$$\frac{\begin{array}{l} e_0,\sigma \rightsquigarrow_P \iota,\sigma_0 \\ e_1,\sigma_0 \rightsquigarrow_P v_1,\sigma_1 \\ \sigma_1(\iota) = [\![\ldots]\!]^c \\ \mathcal{M}(P,c,m) = t\ m(t_1\ x)\ \{\ e\ \} \\ \sigma' = \sigma_1[\text{this}\mapsto\iota][x\mapsto v_1] \\ e,\sigma' \rightsquigarrow_P v,\sigma'' \end{array}}{e_0.m(e_1),\sigma \rightsquigarrow_P v,\sigma''[\text{this}\mapsto\sigma(\text{this}),x\mapsto\sigma(x)]}$$

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

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

fld

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

ass

$$\frac{P,\Gamma \vdash x : t \quad P,\Gamma \vdash e : t' \quad P \vdash t' \leq t}{P,\Gamma \vdash x{:=}e : t'}$$

fldAss

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

cond

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

methCall

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

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

instruction ::= inc | pop | store x | load x | if L | halt

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

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

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

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

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

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

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

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