UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BSc Honours Degree in Mathematics and Computer Science Part II
MSci Honours Degree in Mathematics and Computer Science Part II
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
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 C221=MC221

COMPILERS

Monday 26 April 2004, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

Section A (Use a separate answer book for this Section)

1 This question concerns a simple programming language with functions and arithmetic expressions. Programs are represented using a Haskell abstract syntax tree:

```
type Prog = [Function]
data Function = Defun String String Exp
data Exp = Const Int | Var String | Plus Exp Exp | Apply String Exp
```

For example, the Haskell expression

```
[Defun "inc" "x" (Plus (Var "x") (Const 1)),
 Defun "main" "x" (Plus (Const 2) (Apply "inc" (Plus (Const 3)(Var "x"))))]
```

represents this program:

```
inc(x) { return x + 1; }
main(x) { return 2 + inc(3 + x); }
```

Your task is to write a code generator for this language. Your code generator should produce code for a 68000. Instructions are represented in Haskell using this data type:

```
data Instr = Define String          -- "label:"
           | Jsr String             -- jump to subroutine, push PC
           | Ret                    -- return from subroutine, pop PC from stack
           | Mov Operand Operand -- "mov.l xxx yyy" (yyy:=xxx)
           | Add Operand Operand -- "add.l xxx yyy" (yyy:=yyy+xxx)
data Operand = Reg Register -- specifies data or address register
             | Push         -- "-(a7)" (as in "mov.w d0,-(a7)" to push d0)
             | Pop          -- "(a7)+" (so "Mov Pop (Reg D0)" = mov.w (a7)+,d0)
             | ImmNum Int   -- "#n"
data Register = D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | A7
```

a Using Haskell (or pseudocode), write a code generator for `Functions`. Assume the existence of a code generation function for expressions, `transExp`, which you will define shortly. Functions should return their result in register `D1`.

b Write down a code generation function `saveRegs`, which identifies which registers are currently in use, and generates code to push them onto the stack.

c Write down a code generation function `transExp`, which generates code for an expression (`Exp`), given a list of available registers. The expression's result is to be left in the first register in the list. Assume that functions can have just one parameter (`Var "x"`), which is stored in register `D0`. Your code generator need not handle running out of registers, but should use as few as possible.

d Explain why "callee-saves" is sometimes a better strategy than "caller-saves" for saving and restoring registers when a function is called.

The four parts carry, respectively, 25%, 15%, 50%, and 10% of the marks.

2 a What is the definition of a back-edge?

b Draw a control flow graph which contains a cycle, but no back-edge.

c Using suitable pseudo-code, show a simple iterative algorithm to compute the dominators for each node of a control flow graph. Your algorithm should find, for each node n, the set *Doms*(n) of nodes which dominate n.

d Consider the following loop:

```
Line 1:  int f(int i, int j, int A[]) {
     2:    int k,r;
     3:    r = 0;
     4:    while (i<j) {
     5:      k = A[j];
     6:      if (k <= 0) break;
     7:      i = i + k;
     8:      r = r + A[i];
     9:    }
    10:    return r;
    11:  }
```

Can the statement at line 5 be hoisted out of the loop? Explain your answer in terms of reaching definitions and dominators.

The four parts carry, respectively, 10%, 25%, 35%, and 30% of the marks.

Section B (Use a separate answer book for this Section)

3a Give an algorithm for producing an LL(1) parse table for an LL(1) grammar.

b Use your algorithm to construct the LL(1) parse table for the LL(1) grammar:

```
A  →  '-' A | '(' A ')' | V X
X  →  '-' A | ε
V  →  id Z
Z  →  '(' A ')' | ε
```

where `'-'`, `'('`, `')'`, and **`id`** denote tokens (terminals).

Show clearly the FIRST and FOLLOW sets that you use.

c With the aid of a diagram outline the operation of an LL(1) pushdown automaton.

d For the parse table in part b, show a run (i.e. the Stack, Tokens and Action) of the LL(1) automaton for the input:

```
( - id )
```

The four parts carry, respectively, 25%, 25%, 25%, and 25% of the marks.

4a Give an AST class for Java's **return** statement:

```
ReturnStatement → return | return Expression
```

Your class should include syntactic and any semantic attributes as well a `check` method that reports as many semantic errors as possible. You can assume that there is a function `Get_Enclosing_Method` that returns a reference to the method that encloses the return statement, or `null` if there isn't an enclosing method. You should state any additional assumptions that you make.

b Draw a fully labelled diagram showing clearly the memory layout after execution of the following:

```
interface G { method p, method q }
interface H { method r, method q }

class A implements G, H {
     H h
     method m ( ) { ... }
     method n ( ) { ... }
     method p ( ) { ... }
     method q ( ) { ... }
     method r ( ) { ... }
     method s ( ) { ... }
}

class B extends A {
     method m ( ) { ... }
     method p ( ) { ... }
     method r ( ) { ... }
}

A a = new A ( )
B b = new B ( )
a.h = b
b.h = a
```

c What is pointer-reversal marking? Give an example that illustrates the technique.

d Consider a programming language that creates stack frames (activation records) for method calls on the heap rather than on the system stack. Elaborate on how such an approach might be implemented. Discuss the advantages and disadvantages of such an approach.

The four parts carry, respectively, 25%, 25%, 25%, and 25% of the marks.