UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
BSc Honours Degree in Mathematics and Computer Science Part I
MSci Honours Degree in Mathematics and Computer Science Part I
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 Royal College of Science
Associateship of the City and Guilds of London Institute

PAPER 1.2 / MC 1.2

MATHEMATICAL REASONING – PROGRAMMING

Wednesday, April 28th 1999, 4.00 – 5.30

Answer THREE questions

For admin. only:
paper contains 4 questions

1a State the principle of list induction.

b Recall that list concatenation ++ is defined by:

```
++ :: [a] -> [a]
[]++ys        = ys
(x:xs)++ys    = x:(xs++ys)
```

We also define `[x]` to be `x:[]` for any `x : a`.

i) Prove by list induction that for all lists `xs : [a]`,

```
xs++[] = xs.
```

ii) Prove by list induction that ++ is associative.

c Two Haskell functions to do with reversing lists are defined as follows:

```
Rev :: [a] -> [a]
Rev []       = []
Rev x:xs     = (Rev xs) ++ [x]

T :: [a] -> [a] -> [a]
T [] ys      = ys
T x:xs ys    = T xs x:ys
```

i) Using part b, or otherwise, show by list induction on xs that for all lists `xs, ys : [a]`,

```
T xs ys = (Rev xs) ++ ys.
```

Take care to justify each step of your argument.

ii) Deduce that for all lists `xs : [a]`,

```
T xs [] = Rev xs
```

The three parts carry, respectively, 20%, 40%, 40% of the marks.

2 Let `A : array 0.. of int` be a Turing array, and let `i,j` be integers with `0 ≤ i,j ≤ Upper(A)`. Recall that we write `A(i to j)` for the Haskell list `[A(i),A(i+1),...,A(j-1)]`. Note that `A(j)` is not included. Formally,

```
A(i to j)    | i=j        = []
             | i<j        = A(i to j-1) ++ [A(j-1)]
```

Consider the Haskell function `Rot` that cyclically rotates a list:

```
Rot :: [a] -> [a]

Rot []       =  []
Rot x:xs     =   xs ++ [x]
```

In this question you are asked to implement a Turing procedure corresponding to `Rot`:

```
procedure Rotate(var A : array 0.. of int)
%pre: none
%post: A(0 to N) = Rot(A0(0 to N)) where N = upper(A)+1.
```

The idea is to save `A(0)`, and then, in a loop, put `A(1)` into `A(0)`, put `A(2)` into `A(1)`, ..., put `A(i+1)` into `A(i)`, ... Finally, put the saved value of `A(0)` into `A` in the right place.

a i) Draw a diagram of `A`, showing the pointer '`i`', representing the situation at the beginning of an arbitrary cycle in the execution of the loop.

ii) Use it to give a loop invariant saying (among other things) what the list `A(0 to i)` is, in terms of `A0`, at the beginning of this cycle of the loop.

b Write the body of `Rotate`. Include a loop variant.

c Show that the loop code re-establishes the loop invariant. Remember to check that all array accesses are legal.

d Show that the loop terminates, and that when it does, the post-condition is set up.

The four parts carry, respectively, 15%, 30%, 30%, 25% of the marks.

Turn over ...

3a The following Haskell function calculates the sum of the squares of the entries in a list:

```
Sumsq :: [Int] -> Int
Sumsq []        = 0
Sumsq x:xs      = x*x + Sumsq xs
```

i) Write down a tail-recursive Haskell function `TRSumsq` with an accumulating parameter that, when called with suitable arguments, serves to calculate `Sumsq xs`.

ii) Prove by induction that `TRSumsq` does calculate the same result as `Sumsq`. You will have to formulate an inductive hypothesis that handles any value of the accumulating parameter.

b The Fibonacci sequence f(n) for natural numbers n is given by

f(0)=0, f(1)=1, f(2)=1, f(3)=2, f(4)=3, f(5)=5, f(6)=8, ...

Give a recursive Haskell or Turing definition of f.

c A tail-recursive function that can be used to calculate f as in part b is given below:

```
TrFib :: int -> int -> int -> int
||pre:  n>=0
||post: Trfib x y n = x*f(n) + y*f(n+1)

Trfib x y n     | n==0          = y
                | otherwise     = Trfib y x+y n-1
```

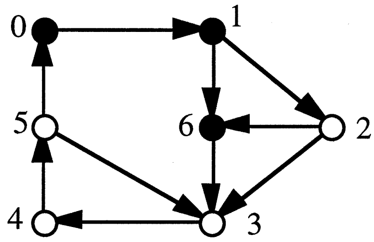
i) Prove by induction on n that `TrFib` meets its post-condition.

ii) What arguments must you give the `TrFib` function to calculate `f(n)`? Justify your answer.

d Write the Turing code of a procedure `LpFib(x,y,n)` that calculates `TrFib` by using a loop, without recursion. Do not forget to include a pre-condition, post-condition, loop variant, and loop invariant as comments.

The four parts carry, respectively, 30%, 15%, 30%, 25% of the marks.

4 A *piebald graph* is a directed graph whose nodes are coloured black or white. E.g.:



A *monochromatic path* between two nodes x, y is a path from x to y, all of whose nodes, including x and y, are the same colour. E.g., above, the path 0,1,6 is a monochromatic path from 0 to 6, because 0, 1, and 6 are black. Similarly, 3,4,5,3 is a monochromatic path from 3 to 3. The path 0,1,2 is not monochromatic, because 0 and 1 are black and 2 is white.

It is desired to adapt Warshall's algorithm to determine which nodes of an arbitrary piebald graph have a monochromatic path between them. The graph has nodes `0,1,...,N` and is represented by the two arrays `Edges` and `Black`, where:

`Edge(x,y)` = **`true`** when there is an edge from `x` to `y`
`Black(x)` = **`true`** when `x` is a black node.

Here is the header:

```
type Adj : array 0..N,0..N of boolean
type Colour : array 0..N of boolean

function MonchromPath(Edge: Adj, Black: Colour) : Adj
% Post: ∀x,y:int[0≤x≤N & 0≤y≤N →
%        (r(x,y) ↔ there is a monochromatic path from x to y)]
```

a Write the code for `MonchromPath` and give a suitable loop invariant and loop variant.

b i) Prove that the initialisation code establishes your loop invariant.

ii) Prove that the loop code re-establishes your invariant.

c An *alternating path* between nodes x, y of a piebald graph is a path from x to y whose successive nodes are of different colours. E.g., in the graph above, 1,2,6,3 is an alternating path from 1 to 3.

How would you modify the code for `MonchromPath` so that it determines which nodes have an alternating path between them? Briefly justify your answer.

The three parts carry, respectively, 30%, 50%, 20% of the marks.

End of paper

