UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1996

MSc Degree in Computing Science
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Diploma of Membership of Imperial College

PAPER M350

ALGORITHMS AND REASONING

Friday, May 17th 1996, 10.00 - 12.00

Answer THREE questions

For admin. only: paper contains
5 questions
4 pages (excluding cover page)

1a i) State the principle of *course of values* induction.

ii) What is a *recursion variant?* Use course of values induction to explain how the existence of a recursion variant helps in proving correctness of recursive functions.

b Suppose we wish to implement a function to list the prime factors of a number n:

```
primes::num -> [num]
||pre: nat(n) & n >= 1
||post: the elements of (primes n) are all prime
||        & their product is n
```

Consider the following auxiliary function:

```
ps::[num]->num->num->[num]
||pre: nat(m) & nat(n)
||  1 <= n, 2 <= m
||  ∀r:nat. (1 < r < m -> r does not divide n)
||post: ps xs m n
||  = (a list satisfying the spec for primes n)++xs

ps xs m n
    = xs,        if n = 1
    = ps (m:xs) m (n div m),   if n mod m = 0
    = ps xs (m+1) n,           otherwise
```

i) Implement `primes` using `ps`. Use the specification of `ps` to prove that `primes` satisfies *its* specification.

ii) Prove that `ps` satisfies its specification, assuming that its recursive calls work correctly.

iii) Justify the assumption in ii) about recursive calls.

The two parts carry, respectively, 40%, 60% of the marks.

2a Specify the following informally described functions by pre- and post-conditions. You may use ++ (list concatenation) and # (list length) within the specifications.

i) isin::*→[*]→bool
||isin x xs = True iff x is an element of xs

ii) sub::num→[*]→*
||sub n xs = nth element of xs (counting the head as element number 0)

iii) nodups::[*]→bool
||nodups xs = True iff xs contains no duplicate values,
|| i.e. no value of type * occurs more than once as element of xs.

b The Merge relation on lists is described informally as follows. Merge(xs,ys,zs) holds iff zs is a merge of xs and ys: the elements of zs are those of xs and of ys interleaved, but the elements of xs preserve their order in zs and so do those of ys. For instance, Merge(['c', 'a', 't'], [3,1,3], [3,1,'c','a',3,'t']).

Specify the following functions using Merge. (You may also use ++, # and isin.)

i) scrub::*→[*]→[*]
||scrub x xs is xs but with all occurrences of x removed

ii) split::(*→bool)→[*]→([*],[*])
||split p xs = (us,vs) where
|| us is the sublist of xs comprising those elements satisfying p
|| vs is the sublist of xs comprising those elements not satisfying p

c Consider the following function:

```
thin::[*]→[*]
||thin xs contains the same elements as xs, but with no duplications
||pre: none
||post: nodups(ys) ∧ ∀x:*. isin(x,xs) ↔ isin(x,ys)
||          where ys = thin xs
thin [] = []
thin (x:xs) = x:(thin (scrub x xs))
```

Using the formal specification of nodups from part a, explain why thin meets the "nodups(ys)" part of its specification.

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

Turn over ...

3a i) What is a *loop invariant?*

ii) How must loop invariants relate to postconditions?

iii) What is a *loop variant,* and what is it used for?

b Consider the following specification:

```
procedure Divs(var A:array[0..N]of integer;
                                     M:integer);
{pre: 0 ≤ M ≤ N}
{post: ∀i:integer.(0 ≤ i ≤ N → A[i] = (M*i)div N)}
```

(Incidentally, a similar calculation is needed when plotting a straight line through a pixel grid.)

It is possible to implement this without ever calling div, using the following idea. If for a given i you know both q = (M*i)div N and r = (M*i)mod N, then you can easily calculate the corresponding values for i+1: add M to r, and iff that makes r ≥ N then subtract N and add 1 to q.

i) Write a Pascal implementation of this, including a loop variant and using the following loop invariant:

{invariant: $1 \le n \le N+1 \wedge \forall i{:}\text{integer}.\ 0 \le i \le n-1 \rightarrow A[i] = (M*i) \text{ div } N$
$\wedge\ q = M*(n-1) \text{ div } N \wedge r = M*(n-1) \text{ mod } N$}

ii) Give a full correctness proof for your implementation.

The two parts carry, respectively, 40%, 60% of the marks.

4 The function Search is specified informally as follows:

```
function Search(A:array[1..N]of integer; x:integer)
                              :integer;
{pre: Sorted(A)}
{post: 0 ≤ result ≤ N
        & result marks the end of the region of A
        in which the elements are ≤ x}
```

Note! The inequality "≤ x" here makes this specification different from the one that was considered in lectures (with "< x").

a i) Write down a formal logical postcondition for Search.

ii) Implement the specification using the binary search algorithm, including loop variant and invariant.

b i) What assumptions are you using about integer division? Explain exactly where these assumptions are used in your implementation.

ii) What can go wrong with this algorithm if integer division is not specified precisely enough?

The two parts carry, respectively, 65%, 35% of the marks.

5a Explain the *partial correctness* interpretation of a Hoare triple P{C}Q, where P and Q are logical formulae and C is a program command.

b State the proof rules in Hoare logic for –

i) assignments "x:=e",

ii) sequential compositions "$C_1;C_2$",

iii) while loops "while B do C".

c Use the rules to show how the triple

$$P\{C_1;\ (\text{while } B \text{ do } C_2);\ C_3\}Q$$

can be proved from three triples of the form $-\{C_i\}-$ ($i = 1, 2, 3$).

d The standard proof rule for "while B do C" assumes that B has no side effects. Suppose you wished to get around this by allowing loop tests of the form C';B – i.e. first execute C' and then evaluate B – with proof rule

$$\frac{I\{C_1\}J \qquad J \wedge B\{C_2\}I}{I\{\text{while } C_1;B \text{ do } C_2\}J \wedge \neg B}$$

Implement this new construction using the standard while loops, and prove that the new proof rule can be derived from the old ones.

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

End of paper

