UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1996

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing 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 City and Guilds of London Institute

PAPER 1.8

MIRANDA AND PROLOG PROGRAMMING

Wednesday, May 1st 1996, 2.00 - 3.30

Answer THREE questions

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

1a For this question all numbers are non-negative integers. In Miranda a list of numbers from 2 to n can be written in shorthand list notation as

```
[2..n]
```

How would you write a function `nums` that took `n` as its parameter and returned this list as its result? The shorthand list notation must not be used.

b Write a function `rid` that given a number `n` and a list of numbers returns a copy of this list with all the multiples of n removed.

c Write a function `remove` that takes a list assumed to be of the form `[2..n]` and returns a copy of this list with all non-prime numbers removed. A number is non-prime if it can be expressed as a multiple of two numbers other than itself and 1. You may use your answers from parts a and b of this question.

d Write a function `primes` which returns all the prime numbers ≤ a given number. You may use your answers from earlier parts of this question.

```
primes 3 = [2,3]

primes 15 = [2,3,5,7,11,13]
```

All functions need to include both declarations and definitions.

2 An arithmetic expression is built out of numbers and the four operations +, -, × and /. The structure of an arithmetic expression is described by the following type definition:

```
aexp     ::= Num num | Exp aexp aop aexp

aop      ::= Add | Sub | Mul | Div
```

a Define a function `eval` of type `aexp→num` to evaluate `aexp`s.

b Assuming that there is a function `show :: *→[char]` which converts a number to a string of characters, define the function `print::aexp→[char]` which prints an arithmetic expression; for example:

```
? print (Exp (Num 42) Add (Num 6))

(42+6)
```

c By analogy with the function `fold` on lists, define the function `foldexp::(num→*)→(*→aop→*→*)→aexp→*`

The intention is that the first argument applies to numbers and the second to arithmetic expressions involving operators.

d Redefine the functions `eval` and `print` using your answer to part (c).

Turn over ...

3a *Note—* throughout this question the *only* Prolog primitives you may use are "member" and "not".

Write a Prolog program defining just *one* relation remdups2(X, Y) which holds when

X is a list, and
Y is the list that would be obtained by removing all duplicates from X, preserving just the last occurrence of each distinct element in X.

Example: the query ?remdups2([1, 2, 1, 2, 3], Y)
should succeed and bind Y to the list [1, 2, 3].

Sketch the evaluation of ?remdups2([1, 2, 1, 2, 3], Y).
You may omit any steps which evaluate Prolog primitives.

b A potentially efficient method of removing duplicates is to extend one's relation with a third argument S (a list) holding, in reverse order of discovery, those distinct elements of X found so far, and testing whether each next element in X already occurs in S.

Thus, write a Prolog program defining just *one* relation remdups3(X, Y, S) which behaves in this way.

Sketch the evaluation of ?remdups3([1, 2, 1, 2, 3], Y, []).
Again, you may omit any steps which evaluate Prolog primitives.

c Explain carefully why the method used in part b is more efficient than the method used in part a when most members of X are duplicates.

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

4a Define the general form and meaning of each of these Prolog primitives.

i forall
ii findall

b A Prolog family database consists of a set of variable-free "child_of" facts.
For example, child_of(chris, amelia) would mean that chris is a child of amelia.
The database is such that no two persons have the same name.

Write a Prolog program which defines the relation most_desc(X) which means that X is a person mentioned in the database and no other person mentioned in the database has more descendants than does X.
You may use freely any of the Prolog primitives.

Hint— one way of solving the problem is roughly as follows:

1. define the relation desc(D, X) meaning D is a descendant of X;
2. find the list L of all pairs (N, Y) such that Y is a person and N is the number of their descendants; this list will include the pair for the person you are looking for;
3. sort L into ascending order;
4. from the result of this, extract the person in the last pair.

c Specify a modest (but not too trivial) "child_of" database. Sketch *in brief* the evaluation of the query ?most_desc(X) using your program.

The three parts carry, respectively, 20%, 60% and 20% of the marks.

End of Paper