UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

BEng Honours Degree in Computing Part III
BEng Honours Degree in Information Systems Engineering Part III
BSc 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 Associateship of the Royal College of Science

PAPER 3.27

THE PRACTICE OF LOGIC PROGRAMMING

Thursday, May 6th 1999, 10.00 – 12.00

Answer THREE questions

For admin. only:
paper contains 4 questions

1a The mean value of a *non-empty* set of numbers $\{a_i \mid i = 1, ..., n\}$ is defined as: $\frac{1}{n}\sum_{i=1}^{n} a_i$

Write a program **mean(L, M)** in Prolog, that is able to calculate the mean value M of a *non-empty* list of numbers L. Your program should call an auxiliary procedure which uses accumulators. Explain carefully how the program works and trace its behaviour when dealing with the query:

```
?- mean([1,3], X)
```

b Write a Prolog program **bin(L, FD)** that creates a frequency distribution FD, expressed as a list of `[value,frequency]` pairs, from a finite list L of numbers. For example, the query:

```
?- bin([2,1,4,3,6,4,3,2,2,4], FD).
```

should yield:

```
FD = [[1,1],[2,3],[3,2],[4,3],[6,1]]
```

The `[value,frequency]` pairs should end up in increasing value order, as in the example. However, try ***not*** to use the library function `sort(L1, L2)`.

c A mode of a finite, *non-empty* list of numbers is a value that occurs most often in the list. Clearly, it is quite possible to have more than one mode.

Write a Prolog program **mode(L, ML, F)** that finds the list ML of modes of a *non-empty* list L of numbers and also gives the frequency F of these modal values. For example, the query:

```
?- mode([2,1,4,3,6,4,3,2,2,4], ML, F).
```

should yield:

```
ML = [2,4]
F = 3
```

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

2 Simulation of queuing activities can be achieved efficiently in Prolog by representing queues as difference list terms of the form:

```
Front - Back
```

where `Back` is a tail of list `Front`.

John forms a queue (of one) outside the theatre, hoping for a late return of tickets. Sue then arrives and joins the queue. Then the threesome Tom, Dick and Harry join the queue. Mavis arrives and jumps the queue, going straight to the front. She then lets her friends, the couple Mike and Jane, go in front of her at the head of the queue (Mike in front). One theatre ticket has been returned, and the person at the front of the queue is served and so leaves the queue.

The state of the queue, Q6, at the end of the story can be found by running:

```
? - make_empty_queue(E),
    join_queue(john, E, Q1),
    join_queue(sue, Q1, Q2),
    list_join_queue([tom,dick,harry], Q2, Q3),
    jump_queue(mavis, Q3, Q4),
    list_jump_queue([mike,jane], Q4, Q5),
    serve_queue(Q5, Served, Q6).
```

a By considering the most general way representing an empty queue, give the Prolog code for:

```
make_empty_queue(E)
```

b Consider the most general way of representing, firstly, a queue containing the single person John and, secondly, a queue containing just John followed by Sue. Then find a new way of representing the first of these queues using the same `Front` component as in the second case. From these considerations, give the prolog code for:

```
join_queue(Element, OldQ, NewQ)
```

```
join_list_queue(List, OldQ, NewQ)
```

without using the conventional `append` program for normal lists.

c Give the Prolog code for:

```
empty_queue(Queue)
```

that ***tests*** whether a queue is empty.

d Give the Prolog code for:

```
jump_queue(Element, OldQ, NewQ)
```

```
list_jump_queue List, OldQ, NewQ)
```

```
serve_queue(OldQ, Served, NewQ)
```

```
length_queue(Queue, Length)
```

without using the conventional `append` program for normal lists.

e Give the Prolog code for

```
append_queues(Q1, Q2, Combined)
```

that represents the action of appending two queues Q1 and Q2 together, so that the members of Q2 join up at the back of Q1 in an orderly fashion? Explain how Prolog would handle the query:

```
?- append_queues([1,2,3|X] - X, [4,5|Y] - Y, Q3)
```

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

[Turn over ...

3a Simple arithmetic expressions can be written in tree form using symbolic function names such as:

```
minus, add, sub, and mult
```

where `minus` is unary and `sub` is binary.

For example, we can write $3x + 2$ as:

```
add(mult(3, x), 2)
```

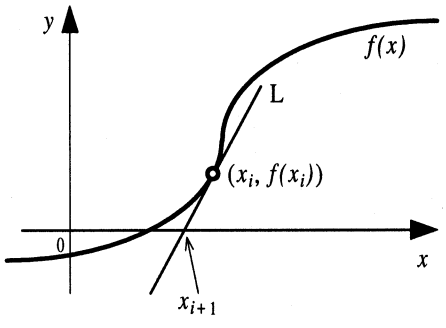
(Note that the atom x represents an algebraic variable here).

Making use of Monash CLP(R)'s built-in library function `arithmetic(X)` which succeeds if X is a variable-free arithmetic expression, write a program for **`evalsimp(E, X, Y)`** that takes the tree representation E of a simple expression in algebraic variable x, involving just the +, - and * operators, and returns the value Y of that expression for a given value X of x.

b Carefully trace how Monash CLP(R) handles the query:

```
?- evalsimp(mult(x, add(x, 1)), 2, V).
```

c The Newton-Raphson iterative method of equation solving starts from an initial guess x_0 for the root of the equation $f(x) = 0$ and repeatedly computes a new guess x_{i+1} of the solution from the previous solution x_i . The new solution is computed by approximating f by a straight line L which passes through the point $(x_i, f(x_i))$ and which has gradient $f'(x_i)$, where f' is the derivative of f. The value of x_{i+1} is the x-intercept of line L.



The process terminates when x_i is sufficiently close to a solution of $f(x) = 0$, i.e., when $-\varepsilon < f(x_i) < \varepsilon$, for some given small tolerance $\varepsilon > 0$.

Assume that there are built-in functions for

- **`evaln(E, X, Y)`**, that is similar to `evalsimp` that you defined in part a, but able to handle any polynomial, and
- **`deriv(F, DF)`** that takes a given polynomial F over x (in tree form) and computes its derivative DF with respect to x (again in tree form).

Give the CLP(R) program for **`nrsolve(F, X0, EPS, X)`** that takes a polynomial F over x (in tree form), an initial guess value X0, and a very small positive tolerance EPS, and yields a solution X to the equation F(x) = 0.

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

4a Write a general purpose recursive Prolog program for **`item(N, L, E)`**, meaning that E is the Nth element of *non-empty* list L. Your program should be able find the nth element as well as check where an element comes on a list. It should be able to handle lists of any length and should therefore ***not*** simply be a set of assertions, dealing with lists of length 1, 2, etc.

b Describe and explain the role of the main components of a typical "constrain and generate" CLP(FD) program for solving a combinatorial problem.

c Over five consecutive weeks of a recent run of the television quiz show "Holiday of a Lifetime", five contestants each reached the final round and won a big prize, by answering six questions correctly within 50 seconds, actually each in a different time.

All contestants finished on a multiple of eight seconds, but no one took 40 seconds. The first week's winner was eight seconds slower than the Acapulco victor, but quicker than Frederic, who, in turn, appeared later in the series than week 2. Graham took half the time in his show than the winner of the holiday to the Bahamas, and appeared earlier than Hilda. Ian took only eight seconds but did not win the Corfu holiday, whose winner was twice as quick as week 2's success. The winner of the trip to Dominica appeared more than three shows before Janet, who took half the time of the Egypt holiday winner. Week 3's winner took 32 seconds.

Give a complete CLP(FD) solution to the above brainteaser making use of **`item`** defined in part a. You do ***not*** need to include explicit output routines. Rather, the solution values should become available via the arguments of the head of the main program (viz. top(X1, X2, ...) :- ...).

You are not required to find a solution to this brainteaser yourself, but rather to provide the CLP(FD) code that would do so.

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

[End of paper