UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

BEng Honours Degree in Computing 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

PAPER C327

THE PRACTICE OF LOGIC PROGRAMMING

Thursday 6 May 2004, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1a Define, in Prolog, a predicate

```
factorial(int, fact)
```

meaning that `int` is a non-negative integer and `fact` a positive integer whose value is the factorial of `int`. Your program should be entirely flexible in the way that it can be used both to test and to generate solutions, and should fail gracefully, when appropriate, rather than entering an endless loop.

b A Hamming Number is a positive integer which has no prime factor other than 2, 3 and 5. For example, $12 = 2 \times 2 \times 3$ is such a number, but 14 (which is divisible by 7) is not. The problem is to find an efficient way to generate all the Hamming Numbers below 100 in increasing order. It helps to recognise that Hamming Numbers can be defined by the properties:

- 1 is a Hamming Number;
- if H is a Hamming Number, then so are 2×H, 3×H and 5×H; and
- there are no other Hamming Numbers.

i) Implement the following algorithm for generating the Hamming Numbers below 100 in Prolog, using `append` to achieve step D below:

A. Start with three lists, each containing just the number 1.

B. Find the smallest of the three numbers on the front of the lists. If it is below 100, output it as the next Hamming Number, H. Otherwise stop.

C. Remove any occurrences of H from each of the three lists.

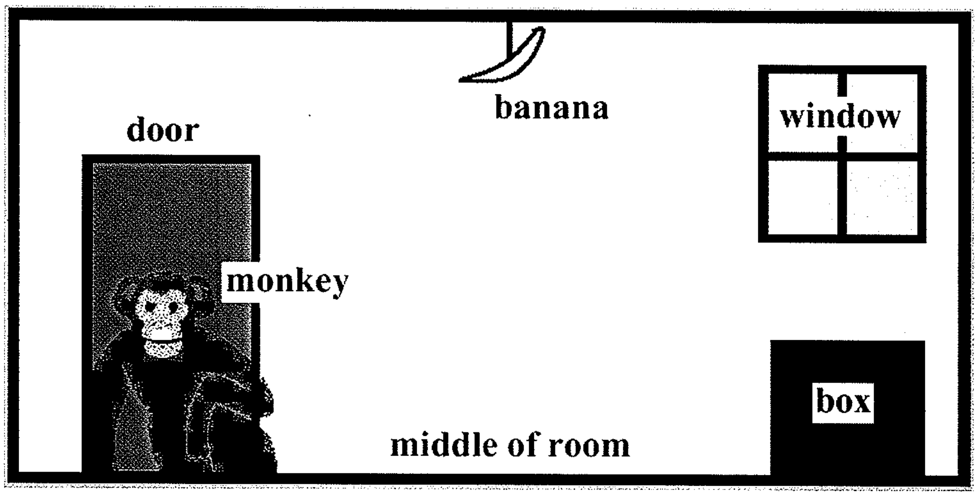
D. Add 2×H to the end of the first list, 3×H to the end of the second list and 5×H to the end of the third list. Return to step B.

You will need a main recursive program that generates and outputs the Hamming Numbers one-by-one, and lower-level clauses that find the smallest of the three numbers on the front of the lists and that remove the latest Hamming Number from the three lists.

ii) Transform the main program from subpart (i), by expressing the lists in difference-list form and then unfolding the occurrences of the difference-list form of `append` with respect to its definition. You may need to slightly adjust the lower-level clauses to conform with changes to your main program.

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

2 A monkey is in a laboratory with a banana hanging out of reach from the ceiling. A box is available that will enable the monkey to reach the banana if he climbs onto it. Initially, the monkey is at the door, the banana in the middle of the room and the box by the window. The monkey and the box have height *low*, but if the monkey climbs on top of the box, he will have height *high*, the same as the banana.



The actions available to the monkey include *go* from one place to another, *push* an object along the floor, from one place to another, *climbUp* onto and *climbDown* from the box, and *grasp* an object. An object can be got by grasping for it, which can be successfully achieved provided the monkey and object are at the same place at the same height. The monkey's problem is how to get the banana.

a Give the domain-independent Prolog clauses (for `holds`, `poss_act`, etc.) that support finding a solution to a planning problem based on its situation-calculus representation.

b Represent the monkey-and-banana world in the situation-calculus, giving Prolog clauses that express facts true at the start, facts that need to hold in the goal state, facts established and facts invalidated by actions and the preconditions of actions.

c What query could you pose that would produce a solution to the problem and how might you arrange for the solution plan to be "pretty-printed"?

The three parts carry, respectively, 40%, 45% and 15% of the marks.

3a Describe the use of consistency technique employed by CLP(FD) in constraint propagation.

b Write a program in CLP(FD) to solve the following problem:

Last week there was a special series of lectures, one each day (Monday to Friday), held in the Clore Lecture Theatre. The subjects were Avionics, Bioinformatics, Cyphers, Databases and Ergonomics. The lecturers were two women named Fanny and Gertrude, and three men named Harry, Ivor, and Jack. Their last names were Kelly, Lewis, Mills, Newman and Osbourne. The problem is to find the full name of each day's lecturer together with the subject, from the following clues:

I. Fanny lectured on Monday.

II. Harry's lecture on Bioinformatics was not given on Friday.

III. Osbourne gave the lecture on Databases.

IV. A man gave the lecture on Cyphers.

V. Ms. Newman and the lecturer on Ergonomics spoke on consecutive days, in one order or the other.

VI. Mills gave a lecture after Jack did.

VII. Ivor Kelly gave his lecture sometime before the Cyphers lecture.

The solution should be output in a meaningful form such as:

```
Monday:    Fanny Newman lectured on Avionics
Tuesday:   Ivor Kelly lectured on Ergonomics
Wednesday: Jack Lewis lectured on Cyphers
Thursday:  Harry Mills lectured on Bioinformatics
Friday:    Gertrude Osbourne lectured on Databases
```

To save time you may use suitable abbreviations for the various names and subjects.

The two parts carry, respectively, 30% and 70% of the marks.

4a Express, as a set of Prolog grammar rules, the word-list form of all positive integers below one thousand. Examples of such positive integer word-lists are:

```
[two],
[fifty, eight] and
[one, hundred, and, twenty].
```

You may omit some rules and simply write "etc.", where you believe that these rules closely resemble earlier ones. However, ensure that you give sufficient details so that the full set of grammar rules could be constructed by following your examples.

b Extend your code from part (a) to produce a program `posint(num, wl)`, in Prolog, that can translate in either direction between the usual numeral form, `num`, and the word-list form, `wl`, of positive integers below one thousand.

c Using the program from part (b), define, in Prolog, a predicate

```
double(int, doubleInt)
```

meaning that `int` is the word-list form of a positive integer under one thousand and `doubleInt` the word-list form of **an even** positive integer under one thousand whose value is twice that of `int`.

d Using a modified version of the program from part (b), define, in Prolog, a predicate

```
double1(num, doubleNum)
```

meaning that `num` is either the word-list form of a positive integer under one thousand, or the list `[a, half]` or the word-list form of a positive integer under one thousand appended to the front of the list `[and, a, half]`, and `doubleNum`. is the word-list form of a positive integer under one thousand whose value is twice that of `num`.

So, for example, the query:

```
?- double1(N, [forty, three]).
```

should yield the answer:

```
N = [twenty, one, and, a, half]
```

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