UNIVERSITY OF LONDON

GOLDSMITHS COLLEGE

B. Sc. Examination 2003

COMPUTING AND INFORMATION SYSTEMS

CS52008A (CS218) DECLARATIVE PROGRAMMING

Duration: 2 hours and 15 minutes

Date and time:

---

*Answer **FOUR** questions.*

*Full marks will be awarded for complete answers to **FOUR** questions.*

There are 100 marks on this paper.

Electronic calculators may be used. The make and model should be specified on the script and the calculator must not be programmed prior to the examination.

This examination paper must not be removed from the examination room.

Question 1

(a) Express the following lists in terms of :: and nil in Standard ML.

(i) [1]
(ii) [[1]]
(iii) [1, 2] [6]

(b) Define a Standard ML function *product* that takes two integers and returns their product. Thus, for example, product(2, 5) should return 10. [2]

(c) Define a Standard ML function *empty* that takes a list and returns *true* if and only if the list is empty. [2]

(d) Define a Standard ML function *length* that takes a list and returns its length. For example, length([4, 5, 6]) should return 3. [3]

(e) The function *f* is defined as follows:

```
fun f(nil) = 0 |
    f(h::t) = 3*h + f(t);
```

Give a step-by-step evaluation of f([1, 2, 3]). [5]

(f) Define a Standard ML function *powerL* that takes a list of integers and change each integer x in the list into x^x. For example, powerL([1, 2, 3]) would return [1, 4, 27], which is the result of $[1^1, 2^2, 3^3]$. **Hint:** you may first define a function *power* that takes two integers u and v and returns u raised to the power of v. [7]

Question 2

(a) Give a step-by-step evaluation of the following expressions in Standard ML:

(i) 2*5-3+4
(ii) 2+4*5=7*3
(iii) if 3*5+1=16 then 1+2*3 else 2*3+4 [6]

(b) Define a Standard ML function *equal* that takes two integers and returns *true* if and only if two integers are equal. [2]

(c) Define a Standard ML function *opposite* that takes an integer list and turns each integer in the list into its negative counter part. For example, opposite([1, 2, 3]) should return [~1, ~2, ~3]. [3]

(d) (i) Define a Standard ML function *found* that takes an integer and a list and determines whether the integer is found in the list. For example, found(2, [1, 2, 3]) should return *true* while found(4, [1, 2, 3]) should return *false*. [2]

(ii) Having defined *found*, give the step-by-step evaluation of the expression:

found(3, [2, 3, 1, 4]) [4]

(e) Define a Standard ML function *ordered* that takes an integer list and returns *true* if and only if the values within the list are in ascending order. For example, ordered([1, 2, 3]) should return *true* while ordered([1, 3, 2] should return *false*. [4]

(f) Define a Standard ML function *get_element* which takes an integer list *l* and an integer *n* and returns the n^{th} element in *l*. For example, get_element([1, 5, 2, 8], 2) should return 5 and get_element([1, 5, 2, 8], 3) should return 2. (Assume that *l* is non-empty and that *n* is not greater than the length of *l*.) [4]

Question 3

(a) What is meant by the term 'constructors' in Standard ML? Give the constructors for the *bool* type, and for the *list* type. [5]

(b) (i) Explain the rules of *empty* and *add* in the following definition of a datatype, illustrating your answer by showing how such a structure containing the numbers 1, 2, and 3 could be represented:

datatype set = empty | add of int * set; [4]

(ii) Define a Standard ML function *sum* that takes a set of integers and returns the sum of the integers in it. [4]

(c) Define a Standard ML function *less* that takes two integers and returns the smaller of these two values. For example, less(2, 3) should return 2. [2]

(d) Define a Standard ML function *all_zeros* that takes a list of integers and returns *true* if and only if every integer in the list is zero. [4]

(e) Write brief notes on Polymorphism and Overloading, explaining the differences between them using *append* and < as examples. [6]

Question 4

(a) What does it mean for two Prolog terms to match? In your explanation **give** the rules for matching in Prolog. [4]

(b) Determine the results of the following queries in Prolog. Explain your answers.

```
?- admires(john, X) = hates(Y, mary).
?- likes([pat, sue], [tom, jim]) = likes(X, [Y, Z]).
```

[4]

(c) Define left-recursion and explain the problem it can cause. **Illustrate** your answer with an example. [5]

(d) Define a predicate *in* with two arguments that returns *Yes* if and only if the first argument is contained in the list that forms the second argument. For example, in(a, [b, a, c]) should return *Yes*. [3]

(e) Suppose the following have been given:

1) male(john).
2) male(steve).
3) female(mary).
4) married(john).
5) married(mary).
6) unmarried(steve).
7) bachelor(X):- male(X), unmarried(X).

Give a step-by-step evaluation of the following queries in terms of unification and goal replacement:

?- married(X). [3]
?- bachelor(X). [6]

Question 5

(a) For each of the following lists, represent the list using the dot functor:

(i) [a]
(ii) [[a]]
(iii) [a, b] [6]

(b) (i) Using predicates *is_city*, *is_beautiful*, and *is_beautiful_city* write Prolog rules and facts that state:

- London is a city.
- London is beautiful.
- If something is a city and is beautiful then it is a beautiful city. [3]

(ii) Illustrate your answer by explaining the execution of a query that asks 'Is London a beautiful city?'. [3]

(c) Explain the role of *backtracking* in Prolog. [3]

(d) Suppose the following Prolog definition has been provided:

```
f([H|_]):- H<5.
f([_|T]):- f(T).
```

Determine the results of the following queries and give brief explanation to your answers:

(i) f([7, 8, 9]) [3]
(ii) f([7, 2, 9]) [3]

(e) Define a Prolog predicate *remove_last* that takes to two lists and returns *Yes* if and only if the second list is the result of removing the last element from the first list. For example, remove_last([a, b, c, d], [a, b, c]) should return *Yes*.

[4]

Question 6

(a) Define a Prolog predicate *same* that takes two arguments returns *Yes* if and only if the two arguments are the same. [2]

(b) Define a Prolog predicate *sum* that takes a list *L* of integers and another integer *N* and returns *Yes* if and only if *N* is the sum of all the integers in *L*. For example, sum([1, 2, 3], 6) should return *Yes* while sum([1, 2, 3], 10) should return *No.* [3]

(c) Define a Prolog predicate *sum2* that adds up all the **even** integers in a list. For example, sum2([1, 2, 3, 4], 6) should return *Yes* while sum2([1, 2, 3, 4], 10) should return *No.* [3]

(d) Define a predicate *member* with two arguments that returns *Yes* if and only if the first argument is contained in the list that forms the second argument. [2]

(ii) Using the predicate *member*, or otherwise, define a predicate *at_least_two* that takes an element *X* and a list *L* and returns *Yes* if and only if there are at least two occurrences of *X* in *L*. For example, at_least_two(a, [a, c, a, d]) should return *Yes* while at_least_two(c, [a, c, a, d]) should return *No*. [4]

(e) Every letter has an ASCII code. For example, the ASCII codes for A, B, a, b are: 65, 66, 97, 98 respectively. The difference between the ASCII code for a lower-case letter and that for the corresponding upper-case letter is 32, as can be seen from the above example.

In Prolog, there is a built-in predicate *name*, which takes an atom *A* and a list *L* and returns *Yes* if and only if *L* is the list of ASCII codes of the characters in *A*. For example, name('AB', [65, 66]) and name(ab, [97, 98]) both return *Yes*. This predicate can be used to turn an atom into a list of ASCII codes, it can also be used to turn a list of ASCII codes into an atom. (**Note:** atoms in lower-case need not be included in single quotes.)

Now, define a Prolog predicate *change_case* that turns an atom in lower-case into its corresponding atom in upper-case. For example, change_case(abc, 'ABC') should return *Yes* while change_case(abc, 'AbC') should return *No*.
[7]

(f) Write brief notes on the **cut** in Prolog, illustrating your answer with an example.
[4]