UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL/MAY 2004 EXAMINATIONS

CSC 324H1 S
St. George Campus

Duration — 3 hours

No Aids Allowed

Student Number: _ _ _ _ _ _ _ _ _ _

Last Name: _____

First Name: _____

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This final examination consists of 12 questions on 16 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.* Answer each question directly on the examination paper, in the space provided, and *use the reverse side of the pages for rough work.* (If you need more space for one of your solutions, use the reverse side of the page and indicate **clearly** which part of your work should be marked.)

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

General Hint: We were careful to leave ample space on the examination paper to answer each question, so if you find yourself using much more room than what is available, you're probably missing something. Also, remember that hints are just hints: you are not required to follow them!

Good Luck!

# 1:	_____/ 10
# 2:	_____/ 8
# 3:	_____/ 10
# 4:	_____/ 4
# 5:	_____/ 10
# 6:	_____/ 8
# 7:	_____/ 3
# 8:	_____/ 4
# 9:	_____/ 10
# 10:	_____/ 6
# 11:	_____/ 10
# 12:	_____/ 17
TOTAL:	_____/100

Question 1. Length in Prolog [10 MARKS]

In this question you will provide two different definitions for the Prolog predicate mylength/2 that computes the length of a list. For example,

```
mylength([a,b,c],X).
X=3

mylength([],Y).
Y=0

mylength([1,[2,3],4],Z).
Z=3
```

(a) [2 MARKS] Define the Prolog predicate mylength/2, without the use of an accumulator.

(b) [6 MARKS] Now redefine the Prolog predicate mylength/2, by defining a helper predicate mylengthacc/3, that uses an accumulator.

```
mylength(List, N) :- mylengthacc(List, 0, N).
```

(c) [2 MARKS] What is the benefit of using the accumulator in defining mylength? What is the benefit of using an accumulator in general?

Question 2. Prolog Queries and Searching [8 MARKS]

Assume that the following database has been loaded into the Prolog interpreter:

```
position(sundin,centre).
position(kidd,goalie).

allPlayers1([]).
allPlayers1([F]) :- position(F,X).
allPlayers1([F|R]) :- position(F,X), allPlayers1(R).

allPlayers2([F]) :- position(F,X).
allPlayers2([F|R]) :- allPlayers2(R), position(F,X).
```

Show the first four answers for each of the following queries. (You may find it helpful to sketch a search tree for your own use to keep track of the computation.)

```
| ?- allPlayers1(X).

     X = _____;

     X = _____;

     X = _____;

     X = _____
```

```
| ?- allPlayers2(X).

     X = _____;

     X = _____;

     X = _____;

     X = _____
```

Question 3. Unification [10 MARKS]

What is the result of unifying the following pairs of expressions? Write the answer in the appropriate box, or "cannot" if the two expressions cannot be unified.

Expression 1	Expression 2	Unification result
`q(X, f(a), g(Y))`	`q(h(b), Z, g(c))`	
`q(X, f(X), g(Y))`	`q(Z, f(a), g(c))`	
`q(X, f(g(Y)), Y)`	`q(Z, f(Z), h(b))`	
`[X,f(a),Y]`	`[X,Z|Y]`	
`[[this,is]|X]`	`[[Y|Z],a,list]`	

Question 4. Prolog Cut [4 MARKS]

Consider the following family relations database.

```
isaMother(X) :- female(X), parent(X,_).
isaFather(X) :- male(X), parent(X,_).

parent(kyros,zoe).              female(lucy).
parent(sarah,zoe).              female(sarah).
parent(lucy,alex).              female(diane).
parent(lucy,daniel).            male(kyros).
parent(richard,alex).           male(richard).
parent(richard,daniel).         male(tom).
parent(tom,will).
parent(diane,will).
```

Unfortunately, the predicate `isaFather(X)` produces duplicate answers. E.g.,

```
?- isaFather(X).
X = kyros ;
X = richard ;
X = richard ;
X = tom ;
No
```

Rewrite `isaFather`, using cuts and helper predicates if you need them, so that each possible answer is returned exactly once.

Please use the back of one of the test sheets for scratch work and show your **final answer** here:

Question 5. Prolog `flip` Predicates [10 MARKS]

Define Prolog predicates corresponding to the following operations on lists:

(a) [4 MARKS] Write the `flip2(L1,L2)` predicate, which holds when `L1` and `L2` are lists which contain the same elements, but in "alternating" orders; it "transforms"

`L1 = [`E_1`, `E_2`, `E_3`, `E_4`, ... `E_{n-1}`, `E_n`]`

into

`L1 = [`E_2`, `E_1`, `E_4`, `E_3`, ... `E_n`, `E_{n-1}`]`.

That is, `flip2` "forms" the second list by re-arranging the elements of the first list — exchanging the first element with the second, the third with the fourth, etc. If `L1` has an odd number of elements, then `L1`'s final element "remains in place" in `L2`.
The following statements hold, for example:

```
flip2( [1,2,3,4], [2,1,4,3] ).
flip2( [a,b,c],   [b,a,c]   ).
flip2( [[a,b],c], [c,[a,b]] ).
```

(b) [6 MARKS] Write a "deep" form of this predicate, `dflip2`, which exchanges alternate elements at each level. The following statements hold, for example:

```
dflip2( [[a,b],[c]],                      [[c],[b,a]] ).
dflip2( [ [1,2], [[[3,4,5]],a,b], c, d ], [ [a,[[4,3,5]],b], [2,1], d, c ] ).
```

(Hint: You may find it useful to use the built-in Prolog predicate `is_list/1` (e.g., `is_list(X)`) which succeeds if `X` is a list, and fails otherwise.)

Question 6. Regular Expressions and CFGs [8 MARKS]

(a) [3 MARKS] Write a regular expression for the following language or if it cannot be described with a regular expression, write "cannot".
All strings of `a`'s and `b`'s that contain the substring *aaa* exactly once.

(b) [5 MARKS] Write a regular expression and CFG for the following language, or if it cannot be described by a regular expression or by a CFG, write "cannot".
All strings of `a`'s and `b`'s that contain n `a`'s followed by m `b`'s, where $n >= 2m >= 0$. This language can be described as $a^n b^m$, $n >= 2m >= 0$. Examples: `aaaabb, aaab, aaaab, a`.

Regular expression:

Context-Free Grammar:

Question 7. Scheme Mystery Function [3 MARKS]

Consider the following Scheme function `what`:

```
(define (what x y z)
   (cond ((null? x) 0)
         ((equal? (car x) z) 0)
         ((equal? (car x) y)
          (+ 1 (what (cdr x) y z)))
         (else (what (cdr x) y z))))
```

Give the result of the following call to `what`:

`(what '(a b a c b a c) 'a 'c)` ANSWER:

Briefly and precisely describe what `what` does:

Question 8. List Manipulation in Scheme [4 MARKS]

Each of the following expressions has a missing piece. Give the name of a single Scheme function that completes the expression, making it have the value shown; or write "cannot" if no Scheme built-in function causes the expression to have the value shown.

Expression	Desired value	function name that completes the expression (or "cannot")
`( '(a b) '(c d))`	`((a b) (c d))`	
`( '(a b) '(c d))`	`((a b) c d)`	
`( '(a b) '(c d))`	`(a b (c d))`	
`( '(a b) '(c d))`	`(a b c d)`	

Question 9. Deep Mapping in Scheme [10 MARKS]

Using Scheme, define a `map`-like function called `gmap`. If `F` is a unary function (a function taking one argument) and `Exp` is a number or a nested list of numbers, then `(gmap F Exp)` is derived from `Exp` by replacing every number, `N`, by `(F N)`. For example,

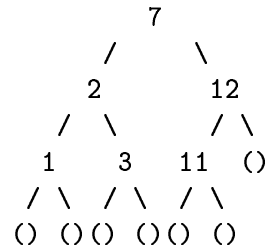
```
(gmap zero? '(0 1 0 2 3))  =>  (#t #f #t #f #f)

(gmap (lambda (X) (+ 1 X)) '(1 (2 3) (4 (5) 6)))  =>  (2 (3 4) (5 (6) 7))

(gmap (lambda (X) (+ 1 X)) '())  =>  ()

(gmap (lambda (X) (* X X)) '5)  =>  25
```

(a) [4 MARKS] Write the function `gmap` in Scheme, but do *not* use `map` in your definition.

(b) [6 MARKS] Write the function `gmap` in Scheme and use `map` to simplify your definition.

Question 10. Trees in Scheme [6 MARKS]

A binary tree is a data structure consisting of nodes and two branches leading to the left and right subtrees. Each node has exactly one *value* and two branches, *left* and *right*. (Note that unlike in class, we are not associating a key with the node, as well as a value.) In this question, we represent a binary tree in Scheme as a list. Empty subtrees are represented by empty lists. For example, the list `(7 (2 (1 () ()) (3 () ())) (12 (11 () ()) ()))` represents the following tree:



Write a recursive Scheme function `(fringe tree)` that takes a tree and returns a list with all leaf nodes (nodes with both subtrees empty). For example,

```
1 ]=> (fringe '(7 (2 (1 () ()) (3 () ())) (12 (11 () ()) ())))

;Value 1: (1 3 11)
```

(Recall ";Value 1: " is what the Scheme interpreter prints out before providing the value of the function, in this case (1 3 11), the fringe of the tree.)
Use `(value tree)`, `(left tree)`, and `(right tree)` as helper functions. These functions are defined as follows:

```
(define (value tree) (car tree))

(define (left tree) (cadr tree))

(define (right tree) (caddr tree))
```

Question 11. Scoping Rules [10 MARKS]

Consider the following program in a Pascal-like programming language:

```
program BIGSUB;
  var: a, b, c, d : integer;
  procedure A;
     var b,c : integer;
     procedure B:
     begin    {B}
       write a,b,c,d;        <-- WHEN YOU GET HERE
     end;     {B}
     procedure C;
       var c, d: integer;
     begin    {C}
       ...
       c:=8; d:=9;
       B;
       ...
     end;     {C}
  begin     {A}
    ...
    b:=6; c:=7;
    C;
    ...
  end;      {A}
begin  {BIGSUB}
  ...
  a:=1; b:=2; c:=3; d:=4;
  A;
  ...
end;   {BIGSUB}
```

(a) [4 MARKS]
Show what the runtime stack would look like the first time you reach the statement labeled "WHEN YOU GET HERE". Show the stack frames including all the control links (i.e., dynamic link to the caller), access links (i.e., static link), and local variable declarations.

To do this, fill in the blanks in the runtime stack on the next page. Fill in the names of the procedures, the control links, the access links, and the local variable declarations. Fill in not only the names of the variables, but also their values (example: a=1, b=2, c=3, d=4). Note that the stack starts at the top and goes downwards in the diagram, just as we did in class.

```
---------------
RTS
---------------
BIGSUB
local vars:
control link: RTS
access link: null
---------------

local vars:
control link:
access link:
---------------

local vars:
control link:
access link:
-----------------

local vars:
control link:
access link:
-----------------
```

(b) [3 MARKS] What does this program print if the language uses lexical scope? (Hint: use the access link.)

(c) [3 MARKS] What does this program print if the language uses dynamic scope? (Hint: use the control link.)

Question 12. Short Answers [17 MARKS]

(a) [2 MARKS]
How many distinct parse trees can an ambiguous grammar generate for a string which is accepted by the grammar? Be precise.

(b) [3 MARKS] Transform the following rules from EBNF notation into BNF notation (remember that curly brackets in EBNF mean zero or more occurrences of the sequence they contain):

<X> ::= <Y> { , <Y> }
<Y> ::= a | b

(c) [2 MARKS]
Why is there no assignment operation in pure functional programming? Briefly explain.

(d) [2 MARKS] Write the following as Horn clauses, or if it can't be done write "cannot be done".

```
If a person is born in May, June, July or August,
then that person is born in the summer.
```

```
If a person is a vegetarian then that person is an animal-lover
or health-conscious or both.
```

(e) [3 MARKS] Given the following top level definitions:

```
(define a 21)
(define b 100)
```

What is the value of each of the following expressions? Place your answer after the word "ANSWER".

```
(let ( (b 12)
       (c (+ b 7)) )
     (* a b c) )
```

ANSWER:

```
(let* ( (b 12)
        (c (+ b 7)) )
      (* a b c))
```

ANSWER:

(f) [2 MARKS] How many distinct derivations can an unambiguous grammar have?

(g) [3 MARKS] Does dynamic scoping increase or decrease a programming language's readability? Briefly explain why.

Total Marks = 100