UNIVERSITY OF VICTORIA
EXAMINATION DECEMBER 2006
CSc 330 F01

NAME:______________________________ STUDENT NO:______________________________

SIGNATURE:____ SECTION: F01

INSTRUCTORS: Dr. M. Cheng DURATION: 3 Hours

TO BE ANSWERED ON EXAMINATION PAPER. NO CALCULATORS ALLOWED.

STUDENTS MUST COUNT THE NUMBER OF PAGES IN THIS EXAMINATION PAPER BEFORE BEGINNING TO WRITE, AND REPORT ANY DISCREPANCY IMMEDIATELY TO THE INVIGILATOR.

THIS EXAMINATION PAPER HAS 13 PAGES. THERE ARE 6 QUESTIONS. ANSWER ALL QUESTIONS.

For Use of Examiner		Marker
1	/20	
2	/15	
3	/15	
4	/15	
5	/20	
6	/15	
Total	/100	

1. (20%) **Lambda Calculus and Type Inference**

   (a) (5%)
   Reduce the following λ-expression to normal form. Show your steps clearly.

   $$(\lambda c.c\ (\lambda a.\lambda b.b))\ \ ((\lambda x.\lambda y.\lambda z.z\ x\ y)\ \ 1\ \ 2)$$

   (b) (5%)
   Using the lambda expression above, write down a Haskell program which will compute the same answer but without any explicit λs (i.e., no `\x->\y-> ...`).

(c) (5%)
Given the following two Haskell functions,

```
compose f g x =  f(g( x ))

product l1 l2   =
    if l == [] then []
    else (compose (head l1) (head l2)) : (product (tail l1) (tail l2))
```

What are the most general types of `compose` and `product`?

(d) (5%)
Lambda Calculus cannot express recursion directly. However, using a *fixed-point* combinator (e.g., `Y`), one can define a recursive function in terms of `Y`. Show how you can express the above recursive `product` function without recursion using `Y`. Explain your steps.

2. (15%) **Programming Language Concepts**

   (a) (2%) What are the *three* notational attributes of a programming language? Explain their roles in defining a programming language.

   (b) (3%) Which are the *three* main paradigms of programming languages? How do they differ in their computational models? Explain.

   (c) (2%) What is *overloading*? Explain how it differs from polymorphic functions in Haskell (e.g., `map`)?

(d) (3%) For a programming language, e.g., Ada, what are the purposes of a declaration? For example,

```
type Cars is (Ford, Rolls, Jaguar, BMW, Opel, Toyota, Honda);
function horsepower ( c : Cars ) return Integer;
c1, c2 : Cars;
```

Discuss what attributes of declarations are used by a program (or the compiler)? Explain.

(e) (5%) You downloaded a copy of the source code of a free compiler for `C#`, written in `C#`, which translates `C#` into Java VM byte code (`JVM`). You want to experiment with `C#` using this compiler on your Pentium machine; however, you cannot execute this free compiler on your machine. With some searching, you were able to download a copy of a `C#` interpreter written in `JVM`. You already have installed a complete Java development environment, which contains an executable of a `JVM` interpreter for your machine. Explain how you may compile a `C#` program `P` on your machine and execute it. Show all your steps.

3. (15%) **Type Checking and Ada**

   (a) (5%) Discuss the *five* basic type construction methods.

   (b) (4%) What are the main purposes of type checking? Explain.

(c) (6%) Given the following Ada type declarations,

```
subtype Positive is Integer range 1..Integer'Last;
type String is array ( Positive range <> ) of Character;
subtype Line is String(1..256);
procedure Put( s : String );
```

Explain what each of the following Ada declarations/statements means and whether they are *legal*, i.e., do not generate a compile time or runtime error.

i. (2%)

```
subtype Index is Integer range 0..Lines'Last;
```

ii. (2%)

```
type Buffer (Size : Positive) is
    record
       l : Line (1..Size);
    end;
```

iii. (2%)

```
Strange : String(11..20) := "Hello";
Put( Strange );
```

4. (15%) **Functional Programming**

   (a) (5%)
   Write a Haskell function `between n m` which returns a list of integers from `n` to `m` inclusively, e.g., `between 2 5 = [2,3,4,5]` and `between 4 3 = []`.

   (b) (5%)
   Write a Haskell function `delete e l` which returns a list with all occurrences of element `e` in list `l` removed, e.g., `delete 3 [1,3,2,5,3,4] = [1,2,5,4]`.

   (c) (5%)
   Write a Haskell function `diff l1 l2` which returns the difference of list `l1` from list `l2`, i.e., the elements in `l1` are removed from the list `l2`. You may assume that `l2` must contain `l1` as a sublist. For example, `diff [2,3] [1,2,3,4,5] = [1,4,5]`.

5. (20%) **Logic Programming**

   (a) (6%)
   You are given the following predicate:

```
concat( [], Y, Y ).
concat( [U|X], Y, [U|Z] ) :- concat( X, Y, Z ).
```

   Show the entire search space (the AND-OR search tree) with all substitutions for the query:

```
?- concat( [1,2], Y, Z ).
```

(b) (6%) Using the above `concat` predicate only,

- (2%)
  define a predicate `enclose( E, L )` which is true if `E` occurs as the *first* and `last` elements in list `L`.

- (2%)
  define a predicate `member( E, L )` which is true if `E` is a member of the list `L`.

- (2%)
  define a predicate `triple( L1, L2 )` which is true if the list `L1` occurs `three` times consecutively in the list `L2`.

(c) (4%)
Write a predicate `alldiff( L )` which is true if all elements in list `L` are not equal. For example, `alldiff( [1,3,2,1] )` is false, `alldiff( [4,2,3,1] )` is true. You may use "\=" as desired.

(d) (4%)

You are given the following predicate, `kmove( (X1,Y1), (X2,Y2) )` which holds if a Knight can move from the position `(X1,Y1)` to the position `(X2,Y2)` in a single move.

```
   +---+---+---+---+---+---+---+---+
 8 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 7 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 6 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 5 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 4 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 3 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 2 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
 1 |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
     a   b   c   d   e   f   g   h
```

Define a predicate `kpath( L )` which holds if `L` is a *non-cyclic* path for the Knight, where `L` is a list of positions on the chess board. For example, `kpath( [ (a,1), (b,3), (d,2) ] )` is true, while `kpath( [ (a,1), (b,3), (a,1)] )` is false.

6. (15%) **Problem Solving in Prolog and Haskell**

   (a) (8%)

   > "Lisa has two bank accounts. The number of each bank has 4 digits. All digits are different. The first account has *four times* the money in the second account. The numbers in each account are reverse of each other. How much money does Lisa has in the first account?"

   Write a Prolog program which can solve this puzzle. You may assume the existence of the `is` predicate in Prolog, i.e., `X is 1+2*2` is true if `X` is 5. (Hint: You don't need to use the list reversal predicate `reverse`.)

(b) (7%) A Pascal Triangle is sequence of numbers of the form:

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10  10  5  1
1  6  15  20  15  6  1
...
```

We can represent this as a list of lists of the form:

`[ [1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1], ...]`

Write a Haskell function `pascals` which generates this infinite list of lists of Pascal Triangle numbers. (Hint: Row 4 is generated by row 3 `[1,2,1]` by summing two lists `[0,1,2,1]` and `[1,2,1,0]` element-by-element.)

END