UNIVERSITY OF VICTORIA
FINAL EXAM
APRIL 2011

NAME: ______________________________	
STUDENT NUMBER: V00________________	

Course No. and Name:	**CSC 330: Programming Languages**
Section:	A01
CRN:	20884
Instructor:	Dr. Mantis Cheng
Duration:	3 hours

- This exam has a total of 16 pages including this cover page.
- Students must count the number of pages and report any discrepancy *immediately* to the Invigilator.
- This is a **closed** book exam. No aid is allowed.
- There are 4 questions. Answer all questions.
- This exam is to be answered on the paper.

Used by Examiner		Marker
1	/25	
2	/25	
3	/25	
4	/25	
Total	/100	

1. (25%) **Lambda Calculus and Functional Programming**

   (a) (5%) Reduce the following λ-expression to normal form (i.e., contains no more β-redexes). Show your steps clearly.

$$((\lambda x.\lambda y.\lambda z.(xz)(yz))\ (\lambda x.\lambda y.x))\ u\ v$$

(b) (5%) Write a Haskell function `merge l1 l2` which takes two *sorted* lists `l1` and `l2` and returns another sorted list by merging the elements of `l1` and `l2`. For example, `merge [1,5] [2,4,5] => [1,2,4,5,5]`. (Note. The lists `l1` and `l2` don't have to be of the same length. Do not remove any duplicates.)

(c) (5%) Write a Haskell function `partition x l` which returns a pair of lists `(xl, xh)` such that all elements in `xl` are less than or equal to `x` and all elements in `xh` are greater than `x`, e.g., `partition 3 [1,3,5,2,6,4] => ([1,3,2],[5,6,4])`. **Note:** Your solution *must* use the `filter` function below,

```
filter p [] = []
filter p (x:xs) =
     if (p x)
     then x : (filter p xs)
     else filter p xs
```

(d) (5%) Write a *recursive* Haskell function `intersect l1 l2` which returns a list of elements which are in both `l1` and `l2`, e.g., `intersect [1,2,3,4] [3,4,5,6] => [3,4]`. **Note**: You may assume every element in list `l1` or `l2` is unique.

(e) (5%) Given `filter` (defined in Part (1c) on page 3) and `map` defined below:

```
map f []     = []
map f (x:xs) = (f x) : (map f xs)
```

Translate the following list comprehension expression in Haskell using `map` and `filter`.

```
    divisors n = [  d  |  d <- [1..n], n mod d = 0 ]
```

2. (25%) **Imperative Programming Language Concepts**

   (a) (5%) Imperative programming languages, such as Java, C/C++, are all based on *assignment statements*, i.e., updates on a shared global memory. On the other hand, functional (e.g., Haskell) and logic (e.g., Prolog) programming languages don't support assignment statements at all. Discuss the pros and cons of assignment statements as a primary programming concept.

   (b) (5%) LISP, Python and Prolog are examples of *dynamically*-typed programming languages; while Haskell and Pascal are examples of *statically*-typed languages. Without being biased, explain the advantages and disadvantages of both approaches to language design from a user (a programmer) point of view.

(c) (5%) *Activation records* are an essential mechanism in implementing block-structured (i.e., lexically scoped) languages, e.g., Pascal, Java, C/C++. Explain whether such a mechanism is needed in implementing functional languages such as Haskell.

(d) (5%) The following is a *polymorphic* Haskell function:

```
reduce f u [] = u
reduce f u (x:xs) = f x (reduce f u xs)
```

For example,

```
reduce (+) 0 [2,3,4]  => 2 + 3 + 4 + 0   => 9
reduce (*) 1 [2,3,4]  => 2 * 3 * 4 * 1   => 24
```

Given the type system of Haskell is defined by the following syntax:

```
<Type> ::= Integer | <Type> "->" <Type> | <Var> | "[" <Type> "]"
<Var> ::= "'" ( "a" | "b" | "c" | "d" )
```

What is the most general type of `reduce`? **Note**: All built-in functions in Haskell are *curried*.

(e) (5%) Most functional programming languages (including Haskell, Scheme, ML) are implemented by translating its programs into an abstract machine code based on λ-calculus first, and then interpreting the abstract machine code directly or compiling it into native machine code.

Assume that you are given a Haskell compiler, written in Haskell, which translates Haskell programs into λ-code. You're also given a λ-interpreter and a λ-to-C compiler, and both are written in C. C was chosen because every computer installation has a C compiler. Explain how you can bootstrap the Haskell compiler to run on your own computer (say, with an ARM CPU), assuming that you have a C-to-ARM compiler already on your machine.

3. (25%) **Logic Programming and Prolog**

   (a) (5%) You are given the following definition of `member( X, L )` which is true if `X` is an element in list `L`.

```
member( X, [X|Z] ).                            % clause M1
member( X, [U|Z] ) :- member( X, Z ).          % clause M2
```

   Show the entire search space (the derivation tree) with all substitutions for the query:

```
          ?- member( X, [1,2] ).
```

(b) (2%) Given the following predicate `concat( L1, L2, L3 )` which is true if list `L3` is the concatentation of lists `L1` and `L2`.

```
concat( [],     L,  L ).
concat( [X|L1], L2, [X|L3] ) :- concat( L1, L2, L3 ).
```

Write a Prolog predicate `member( X, L )` (as specified in Part (3a) on page 8) in terms of `concat`.

(c) (3%) Write a Prolog predicate `adjacent(U, V, L)` which is true if the elements `U` and `V` are adjacent in list `L`. That is, `adjacent( 2, 3, [1,2,3,4] )` is true and `adjacent( 1, 3, [1,2,3,4] )` is false.

(d) (5%) Using the predicate `concat` as defined in Part (3b) on page 9, what are the answers for each of the following queries, if there is more than one?

- (1%) `?- concat( X, [Y], [1,2,3,4] ).`
- (1%) `?- concat( _, [ U, _, V | _], [1,2,3,4] ).`
- (1%) `?- concat( [ X | Y ], [3,4], [1,2,3,4] ).`
- (1%) `?- concat( [ X | L], [X], [1,2,3,2,1] ).`
- (1%) `?- concat( [1,X], Y, [1,2,3,4] ).`

(e) (5%) The 3 by 3 magic square puzzle was solved in the class. The following is *almost* a solution. **Hint**: You need to define a predicate `alldiff( L )` which is true if all elements in list `L` are different.

```
%  X7, X8, X9
%  X4, X5, X6
%  X1, X2, X3
magic( X1, X2, X3, X4, X5, X6, X7, X8, X9 ) :-
  digit(X1), digit(X2), digit(X3), digit(X4), digit(X5),
  digit(X6), digit(X7), digit(X8), digit(X9),
  S is X7+X8+X9,     % rows
  S is X4+X5+X6,
  S is X1+X2+X3,
  S is X1+X4+X7,     % columns
  S is X2+X5+X8,
  S is X3+X6+X9,
  S is X1+X5+X9,     % diagonals
  S is X3+X5+X7.

digit(1). digit(2). digit(3). digit(4). digit(5).
digit(6). digit(7). digit(8). digit(9).
```

Complete the solution and suggest how you can make it more efficient.

4. (25%) **Concurrent Programming and Concurrency**

   (a) (5%) The following is an implementation of *counting semaphores* in monitors written in Concurrent Pascal.

```
type semaphore = monitor  (* a counting semaphore *)
  var val : integer;  (* value of this semaphore *)
  var nonzero : condition;

  procedure init( n : integer ); (* n must be greater than 0 *)
  begin
    val := n;
  end;

  procedure up;
  begin
    val := val + 1;
    nonzero.signal; (* resume a waiting task *)
  end;

  procedure dn;
  begin
    if val = 0 then nonzero.wait; (* wait for nonzero *)
    val := val - 1;
  end;

begin (* initialization *)
  val := 0;
end;
```

   Modify this implementation so that there is an additional procedure `dn2` which decrements the semaphore by 2, instead of 1. That is, the calling task waits until it can decrement the semaphore by 2.

(b) (5%) Explain in your own words the meaning of the following Path Expressions.

i. `path ( up.dn ) + ( dn.up ) end`

ii. `path 3:(up.dn) end`

iii. `path 3:(up + dn) end`

iv. `path up.( 1:(up + dn) ).dn end`

v. `path ( 2:(up.up) ) + ( 2:(dn.dn) ) end`

(c) (5%) For each of the following Petri net, write down all possible sequences of transitions. (For example, `< a, b, ... >`, `< a, c, ... >`, ..., etc.

i. (2%)

ii. (3%)

(d) (5%) The programming language OCCAM is an implementation of CSP on Transputers. Explain in your own words what each of the following fragments of OCCAM code does. Use a diagram if necessary.

i. (2%)

```
CHAN OF INT  in, out, comm:
PAR
   SEQ
   INT x:
     in ? x
     comm ! x
   SEQ
   INT y:
     comm ? y
     out ! y
```

ii. (3%)

```
CHAN OF INT  in1, in2, c1, c2, out:
PAR
   SEQ
   WHILE TRUE
   INT x:
      in1 ? x
      c1 ! x
   SEQ
   WHILE TRUE
   INT y:
      in2 ? y
      c2 ! y
   ALT
   WHILE TRUE
   INT z:
      c1 ? z
         out ! z
      c2 ? z
         out ! z
```

(e) (5%) Unlike CSP, which uses a *rendezvous* synchronous style of message-passing communication, Erlang uses an *asynchronous* style of message-passing communication. Discuss the pros (advantages) and cons (disadvantages) of CSP-style versus Erlang-style message-passing communication model in terms of ease of use, reliability, and performance issues.

END