Midterm Exam, COMP3031 L1, Fall 2011

Date	Oct 25, 2011 (Tuesday)
Time	19:00-21:00
Instructions:	(a) This exam contains six problems, counting for a total of 100 points.
	(b) Write ALL answers in the exam book. Do not use any other papers.

Name:	Problem	Points
Student ID:	1.	
ITSC Account:	2.	
	3.	
	4.	
	5.	
	6.	

Total:

Problem 1 (15 pts) What is the value of each of the following SML expressions (a)-(c)?

```
(*a*)
fun ip(x,k) : real =
    if k=1 then x
    else if k mod 2 = 0 then   ip(x*x, k div 2)
                        else   x*ip(x*x, k div 2);

(*b*)
fun headc []    = []
  | headc ((h::t) :: L) = h :: headc L;

fun tailc []    = []
  | tailc ((h::t) :: L) = t :: tailc L;

fun tp ([]::L) = []
  | tp L = headc L :: tp (tailc L);

(*c*)
datatype term = atom of string
              | cons1 of term*term
              | cons2 of term*term
              | cons3 of term;

fun et (atom t) = t
  | et (cons1(t1,t2)) = et(t1) ^ et(t2)
  | et (cons2(t1,t2)) = et(t2) ^ et(t1)
  | et (cons3 t) =  "-" ^ et(t);
```

(a)

```
ip(3.0,5);
```

(b)

```
tp [[1,2,3],[4,5,6]];
```

(c)

```
et(cons1(cons2(atom "1", atom "2"),cons1(atom "3", cons3(atom "4"))));
```

Problem 2 (15 pts) What is the type of each of the following SML functions (a)-(c)?

(a)

```
(*foo1*)
fun foo1 x y = x (x (x y));
```

(b)

```
(*foo2*)
fun foo2 x nil = nil | foo2 x (y::z) = (x, y(x)) :: (foo2 x z);
```

(c)

```
(*foo3*)
datatype mycolor = red | yellow | blue;

 fun foo3 nil y z w = nil
   | foo3 (red::x) y z w = (y red) :: (foo3 x y z w)
   | foo3 (yellow::x) y z w = (z yellow) :: (foo3 x y z w)
   | foo3 (blue::x) y z w = (w blue) :: (foo3 x y z w);
```

Problem 3 (20 pts) Write the following SML functions (a)-(d).

(a) `list_merge(L1, L2) = fn : int list * int list -> int list`. Given two lists `L1` and `L2`, both sorted in non-descending order, it returns a list of all elements from `L1` and `L2` in non-descending order, e.g., `list_merge([2,3], [2,4])` returns `[2,2,3,4]`.

(b) `list_distinct L = fn : 'a list -> 'a list`. It returns a list of all **distinct** elements of `L` in the original order. For example, `list_distinct [3,2,1,2]` returns `[3,2,1]`.

(c) `list_count L = fn : 'a list -> ('a * int) list`. If `L` is empty, it returns `nil`; otherwise, it returns a list of tuples with each tuple consisting of a **distinct** element of `L` and the number of occurrences of the element in `L`, e.g., `list_count ["How", "do", "you", "do"]` returns `[("How",1),("do",2),("you",1)]`.

(d) `list_squaresum L = fn : int list -> int`. If `L` is an empty list, it raises an exception `EmptyList`; otherwise, it returns the sum of the squares of all elements of `L`, e.g., `list_squaresum [1,2,3]` returns `14`.

Problem 4 (15 pts) Consider the following grammar in BNF with `<S>` being the starting non-terminal:

```
<S>::= <N><V>
<N>::= a|b|<N>-<V>
<V>::= 0|1<N>
```

(a) Generate all strings of length less than 6 in the language represented by this grammar.

(b) Determine whether string

`b-01a-0`

belongs to the language the grammar generates. If your answer is yes, draw a parse tree for the string. If your answer is no, just say so, and no explanation is needed.

Problem 5 (15 pts) Bitstring expressions are defined as follows:

1. A non-empty string of `0`s and `1`s is a bitstring expression.
2. `not A` is a bitstring expression, given `A` is a bitstring expression.
3. `A and B`, `A or B`, and `A xor B` are all bitstring expressions, given both `A` and `B` are bitstring expressions.
4. Both `A << N` and `A >> N` are bitstring expressions, given `A` is a bitstring expression and `N` number of bits (`N` is `1`, `2`, `3`, `4`, `5`, `6`, `7`, or `8`).

In decreasing order, the precedence of these operators is as follows (the associativity rules are given for the binary operators at the same precedence level):

```
not
>>  <<        (both are left associative)
and  or  xor  (all are left associative)
```

Write an **unambiguous** context-free grammar in BNF for bitstring expressions. Your grammar should conform to the given operator precedence and associativity rules.

Problem 6 (20 pts) Consider the following regular expression:

```
(a+b|ba+)+
```

(a) Generate all strings of length less than 6 using this regular expression.

(b) Write a regular grammar in BNF for the language that the regular expression represents.

Blank Page