CSE 130, Fall 2005: Final Examination

Name: _____

ID: _____

Instructions, etc.

1. Write your answers in the space provided.
2. Wherever it says **explain**, write no more than **three lines** as explanation. The rest will be ignored.
3. The points for each problem are a rough indicator (when converted to minutes), of how long you should take for the problem.
4. Good luck!

1 (15)	
2 (15)	
3 (10)	
4 (20)	
5 (20)	
6 (30)	
7 (15)	
8 (10)	
Total (135)	

1. **[15 Points]** For each of the following Ocaml programs, if the code is well-typed, write down the value of `ans`, otherwise, if the code has a type problem, write "type error".

   (a)
```
let ans =
      let x = 2 in
      let y =
        (let x = 20 in
         x * x) + x
      in
        y * x
```

   (b)
```
let f g x y = g (x + y);;

let g = f (fun x -> List.tl x) 3;;

let ans = g 7;;
```

   (c)
```
let f g x y = g (x + y);

let g = f (fn x => x*x) 3;

let ans = g 7;
```

   (d)
```
let f x y = x + y;

let g = f 10;

let f x y = x * y;

let ans = g 10;
```

2. Consider the following Ocaml function.

```
let rec ru (f,g,base) =
  if (g base) then ru (f,g,(f base))
  else base
```

(a) **[5 Points]** What is the *type* of function `ru` ? Answer this by filling in the blanks:

```
___________ * ____________ * _____________ -> _____________
```

(b) **[10 Points]** Use `ru` to implement a function `reverse : 'a list -> 'a list` that returns the reverse of a list, i.e. `reverse [1,2,3,4]` evaluates to `[4,3,2,1]`, by filling in the blanks below:

```
let reverse l =
   let f ___  = ____________________ in

   let g ___  = ____________________ in

   let base   = ____________________ in

   let (_,r) = ru(f,g,base) in

      r
```

3. **[10 Points]** Two expressions e_1 and e_2 are *semantically equivalent* if in *every* environment E, evaluating e_1 and evaluating e_2 produces the same value. For each of the following pairs of expressions, explain why they are semantically equivalent, or if not, then give an environment that distinguishes the two, i.e. in which evaluating the two expressions gives different results.

(a)

e_1
```
let x = a + 1 in
let y = b + 2 in
  2*x + 3*y
```

e_2
```
let y = b + 2 in
let x = a + 1 in
  3*y + 2*x
```

(b)

e_1
```
  let x = f 0 in
  let y = g x in
  if x > 0 then 0 else y
```

e_2
```
let x = f 0 in
if x > 0 then 0 else g x
```

(c)

e_1
```
(fun a -> fun b -> a * b) a
```

e_2
```
(fun b -> fun a -> b * a) b
```

4. Consider the Ocaml module described below:

```
module Stack : STACKSIG =
  struct
     exception EmptyStack

     type 'a stk = 'a list

     let make x = [x]

     let top l =
       match l with
         [] -> raise EmptyStack
       | (h::t) -> h

     let pop l =
       match l with
         [x] -> (None,[x])
       | (h::t) -> (Some h,t)

     let push (x,s) = x::s
  end
```

and the *two* possible signatures:

(A)

```
module type STACKSIG =
  sig
    type 'a stk = 'a list
    val make : 'a -> 'a stk
    val top  : 'a stk -> 'a
    val pop  : 'a stk -> ('a option * 'a stk)
    val push : 'a * 'a stk -> 'a stk
  end
```

(B)

```
module type STACKSIG =
  sig
    type 'a stk
    val make : 'a -> 'a stk
    val top  : 'a stk -> 'a
    val pop  : 'a stk -> ('a option * 'a stk)
    val push : 'a * 'a stk -> 'a stk
  end
```

(a) [**5 Points**] For which *one* of the signatures (A) or (B), can a *client* can cause the exception `EmptyStack` to get raised ? Write down a client expression that would cause this exception to get raised. For the other signature **explain** why the exception will never get raised.

Signature:

Client Expression:

Explanation:

(b) [**5 Points**] Consider the *client* function:

```
let popall l =
  match l with
    [x] -> []
  | l ->
       (match (Stack.pop l) with
          (None,l') -> []
        | (Some x,l') -> x::(popall l'))
```

For *one* of the signatures (A) or (B), the the client function `popall` compiles, i.e. is well typed. Which one ? What is the inferred type of `popall` using this signature ?

Signature:

Inferred Type: `popall : _____ -> _____`

(c) [**10 Points**] Write an equivalent *tail-recursive* version of `popall` that would compile with *both* signatures.

5. We wish to write an Ocaml program to manipulate *Boolean formulas*. Recall that a boolean formula is one generated by the following grammar:

$$b ::= \quad x \quad | \quad \neg b \quad | \quad b_1 \vee b_2 \quad | \quad b_1 \wedge b_2$$

(a) **[5 Points]** Write an SML datatype `boolexp` to represent boolean expressions by completing the declaration given below:

```
type boolexp = Var of int |
```

Use your datatype, to encode the boolean expression

$$(x_0 \vee \neg x_1) \wedge (x_1 \vee \neg x_2)$$

(b) **[5 Points]** Write a function `eval : bool list * boolexp -> bool` such that: `eval [b_0,b_1,b_2,...] e` evaluates to `true` iff the expression `e` evaluates to true when the variables x_i have the value `b_i`.

(c) **[10 Points]** We would like to print the *truth table* of a boolean expression. Write a function: `inputs : int -> bool list list` that takes an integer as input `n` and returns the list of all possible boolean "inputs" to `eval` of length `n`. Thus, `inputs 2` should evaluate to `[[true,true],[true,false],[false,true],[false,false]` and `inputs 3` should evaluate to:

```
[[true,true,true],[true,true,false],[true,false,true],[true,false,false],
 [false,true,true],[false,true,false],[false,false,true],[false,false,false]]
```

6. For each of the following Python programs, write down the value of `ans`. Write your answers on the blank space on the right.

(a) [**5 Points**]

```
a = 10

def f(a,x):
  a = a + x
  return a

x = f(a,10)
ans = a + x
```

(b) [**5 Points**]

```
a = [10]

def f(a,x):
    a[0] = a[0] + x
    return a[0]

x = f(a,10)
ans = a[0] + x
```

(c) [**8 Points**]

```
class Vector:
  data = []
  def __init__(self,v,n):
     for i in range(n):
      self.data.append(v)

x = Vector(2,2)
y = Vector(3,3)
ans = (x.data,y.data)
```

(d) [**7 Points**]

```
c = [0]

def f(x):
  c[0] += 1
  if x == 0: raise Exception(0)
  r = g(x-1)
  c[0] -= 1
  return g(x-1)

def g(x):
  c[0] += 1
  if x == 0: raise Exception(1)
  r = f(x-1)
  c[0] -= 1
  return r

def do(x):
  try: f(x)
  except Exception,e: return str(e)

r = map(do,[0,1,2,3,4,5,6,7,8,9])
ans = (c[0],r)
```

(e) [**5 Points**]

```
class A:
  def __init__(self):
    pass

  def f(self):
    return "A," + self.g()

class B(A):
  def g(self):
    return ("B")

class C(A):
  y = 0

def do(y):
  try: return y.f()
  except: pass

b = B()
c = C()
ans = (do(b),do(c))
```

7. (a) **[7 Points] Explain** why it is not possible to have Python-style *decorators* in SML.
Hint: It has nothing to do with types.

(b) **[8 Points]** Consider the following implementation of `streamify` for PA4.

```
def streamify(f):
  def g(s):
    for x in s:
      for y in f(x):
        yield y
  return g
```

Give an example function `g`, that:

i. Takes an integer input and
ii. Returns a list of integers as output, such that,
iii. For some input `x`, when `g` is *not decorated* with `streamify` the call `g(x)` *successfully* returns a list, but then `g` is decorated, the call `g([x])` throws a run-time exception.

Function: `g`

```
def g(x):
```

Input:

8. Consider the following Java code:

```
interface A {
  void f(A y);
}

interface C extends A {
  A g(C y);
}

class B implements A {
  int x;
  void f(A y){
    return;
  }
}

class D extends B implements C {

//To be implemented by you




}
```

(a) **[2 Points]** Write *all* the *types* of which `D` is a *subtype*.

(b) **[2 Points]** Write *all* the *classes* from which `D` *inherits*.

(c) **[2 Points]** Does the following method successfully typecheck ? **Explain.**

```
int fc(C c){
  return c.x;
}
```

(d) **[4 Points]** Complete the definition of class `D` so that it successfully typechecks.