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 SML 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)
```
val 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 (fun 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 SML 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 `val 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 rec 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 b -> a * b) a
```

e_2

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

4. Consider the Ocaml structure described below:

```
module Stack : STACKSIG =
  struct
     exception EmptyStack

     type 'a stk = 'a list

     let make x = [x]

     let top = function
       | []     ->  raise EmptyStack
       | (h::t) -> h

     let pop = function
       | [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 rec popall = function
  | [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 a 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 Scala programs, write down the value of `ans`. Write your answers on the blank space on the right.

(a) [**5 Points**]

```
var a = 10

def foo(x: Int) = {
  a = a + x
  a
}

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

(b) [**5 Points**]

```
val a = Array(10)

def f(a: Array[Int], x:Int) = {
  a(0) = a(0) + x
  a(0)
}

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

(c) [**8 Points**]

```
class Vec[A](v:A, n: Int) {
  var data: List[A] = List()
  for (i <- 1 to n) {
      data = data ++ List(v)
  }
}

val x   = new Vec(2, 2)
val y   = new Vec(3, 3)
val ans = (x.data, y.data)

//Hint: List(1,2) ++ List(3, 4) == List(1,2,3,4)
```

(d) [**15 Points**]

```
case class NumEx extends Throwable

var c = 0

def f(x: Int) : Int = {
  c += 1
  if (x == 0) { throw NumEx(0) }
  val r = g(x-1)
  c -= 1
  r
}


def g(x: Int): Int = {
  c += 1
  if (x == 0) { throw NumEx(1) }
  val r = f(x-1)
  c += 1
  r
}

def foo(x: Int) = {
  try { f(x) }
  catch { case NumEx(e) => e }
}

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

(e) [**5 Points**]

```
class A {
  def f : String = "A-" + this.g
  def g : String = throw new Exception
}

class B extends A {
  override def g : String = "B"
}

class C extends A {
  var y = 0
}

def foo(x: A): Any = {
  try { x.f }
  catch {case _ => () }
}

val ans = (foo(new B), foo(new C))
```

7. (a) [**7 Points**] **Explain** why it is not possible to write *decorators* in Ocaml.
Hint: It has nothing to do with types.

(b) [**8 Points**] Consider the following function.

```
def streamify[A, B](f: /* T1 */, xs: /* T2 */) : /* T3 */ = {
  for ( x <- xs
      ; y <- f(x))
  yield y
}
```

Write down appropriate types, such that the function type checks.

- T1
- T2
- T3

(c) Consider the following Java code:

```
trait A {
  def f(y: A) : Unit
}

trait C extends A {
  def g(y: C) : A
}

class B extends A {
  val x : Int = 0
  def f(y: A y) { return }
}

class D extends B with C {

  //To be implemented by you

}
```

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

ii. [**2 Points**] Write *all* the *classes* from which `D` *inherits*.

iii. [**2 Points**] Does the following method successfully typecheck ? **Explain.**

```
def foo(c: C): Int = {
  return c.x
}
```

iv. [**8 Points**] Complete the definition of class `D` so that it successfully typechecks.