Preliminary Examination I
Computer Science 312, Cornell University 11 March 2004

1. True/False [10 pts]

   (parts a–e; **4** points off for each wrong answer, 2 points off for each blank answer, minimum problem score 0.)

a. ____ In a data abstraction, a given abstract value may be represented by more than one concrete value.

b. ____ In SML, unbound variables are always detected at compile time.

c. ____ Polymorphism is the language feature that provides recursive datatypes.

d. ____ The `foldl` function is tail-recursive.

e. ____ Black-box test cases are written without reading the implementation that is being tested.

2. Specifications [20 pts] (parts a–e)

   Consider the following specifications and code for functions named `f` of type `int*int->int`:

(a)
```
(* f(x,y) = x + 1.
 * Requires: x >= 0 *)
```

(b)
```
(* f(x,y) >= 0.
 * Requires: x >= 0 & y >= 0 *)
```

(c)
```
(* f(x,y) = x + y
 * Requires: x >= 0 & y >= 0 *)
```

(d)
```
(* f(x,y) >= 0
 * Checks: x > 0 *)
```

(e)
```
(* f(x,y) = x
 * Requires: x > y *)
```

(1)
```
fun f(x:int, y:int):int =
  Int.abs(x) + 1
```

(2)
```
fun f(x:int, y:int):int =
  x+y
```

(3)
```
fun f(x:int, y:int):int =
  if x < y then y else x
```

(4)
```
fun f(x:int, y:int):int =
  case (x,y) of
    (0,_) => y
  | _ => x
```

(5)
```
fun f(x:int, y:int): int =
  let fun g(z: int) =
    if z = 0
      then x
      else g(z-1)
  in
    g(y)
  end
```

(6)
```
fun f(x:int, y:int):int =
  let val z = (if x < 0
               then 0
               else x)
  in
    y div z
  end
```

For each of the specifications (a)–(e), identify the names of the function declaration or declarations in (1)–(6) that implement it. Some specifications might not be implemented by any function; some might be implemented by more than one function; some functions might implement no specifications or might implement multiple specifications.

(a) [4 pts] ____

(b) [4 pts] ____

(c) [4 pts] ____

(d) [4 pts] ____

(e) [4 pts] ____

3. Evaluation [17 pts] (parts a–c)

   Consider the following function definition:

```
fun f(p, xs: int list) =
  case xs of
    [] => ([], [])
  | x::t =>  let val (ys,zs) = (f(p, t)) in
      if x > p then (ys, x::zs) else (x::ys, zs)
    end
```

   (a) [4 pts] The implementer has not been reading the CS 312 style guide carefully and have left off some type declarations. What are the types of `p`, `ys`, and `zs`, and `f`?

   (b) [5 pts] Given this function definition, what is the value that results from evaluating the expression `f(3, [2,4,1,5])`?

   (c) [8 pts] Write a specification for the function `f` that is deterministic, definitional, and has no precondition. Be concise but accurate.

4. Data Abstraction [33 pts] (parts a–f)

   The following is the interface and implementation of a data abstraction written by someone who missed the lecture on documenting implementations.

```
signature INTERVAL = sig
  (* An interval is a (possibly empty) contiguous set of integers; that is,
   * a (possibly empty) set of n integers [a,b] = {a, a+1, * a+2, ..., b}.
   * (Note that [a,b] is ordinary mathematical notation, NOT an SML list!)
   * Examples: ∅, [0,2], [~2,~1], [~1,102] *)
  type interval
  (* create(x,y) is the set of integers n such that x<=n<=y. *)
  val create: int*int -> interval
  (* top(a) is the largest integer in a. *)
  val top: interval -> int
  (* bottom(b) is the smallest integer in a. *)
  val bottom: interval -> int
  (* The number of elements in the interval. *)
  val size: interval -> int
  (* join(a,b) is the smallest interval including both a and b *)
  val join: interval*interval -> interval
  (* intersect(a,b) is the largest interval included in a and b *)
  val intersect: interval*interval -> interval
end

structure Interval :> INTERVAL = struct
  type interval = int*int
  fun create(a0: int, a1:int): interval = ...
  fun size((a0,a1)) => a1 - a0 + 1
  fun top(a0,a1) = a1
  fun bottom(a0,a1) = a0
  fun intersect((a0,a1), (b0, b1)) =
    create (Int.max(a0,b0), Int.min(a1, b1))
  fun join(a,b) = ...
end
```

(a) [8 pts] Some of these methods need requires clauses. Which are they? For each method, explain why a requires clause is needed, and give a requires clause that addresses the problem you have identified.

(b) [4 pts] The implementer has failed to give an abstraction function and representation invariant. Give a specification of the abstraction function that this implementation uses.

(c) [4 pts] Give a representation invariant that is strong enough to prove that all of the implemented functions work correctly. (Do not give the proofs.) Make sure your rep invariant agrees with the abstraction function given just previously. **Hint:** look at the function `size`.

(d) [6 pts] The `create` function is unimplemented. Write an implementation that meets the specification and ensure the rep invariant.

(e) [6 pts] The `join` function is unimplemented. Write an implementation that satisfies the given specification and preserves the representation invariant. **Hint:** Look at `intersect`, but don't copy it.

(f) [5 pts] One possible test case for the function `join` is `join(`$[1,3]$`, `$[2,4]$`)` (where Roman font is used here to indicate abstract interval values rather than SML lists).

Suggest five more good black-box test cases for `join` that provide as much coverage as you can achieve. For each test case, write three or four words (a full sentence is not necessary) explaining how it improves coverage. You will lose points for providing redundant test cases.

5. Complexity and Induction [20 pts] (parts a–c)

Consider the following function:

```
fun f(n: int): int =
  if n <= 1
    then n
    else f(n div 2) + f((n+1) div 2)
```

(a) [6 pts] Given this function definition, show every step in the evaluation of the expression `f(3)` according to the evaluation model given in class. For each step, put an underscore under the subexpression that is being reduced. For brevity, you may use ellipses (...) to abbreviate unchanged parts of the expression carried over from the previous evaluation step or from the definition of `f`.

(b) [8 pts] Prove using induction that `f(n) = n` for all nonnegative `n`. Be sure to state clearly what you are proving and what your induction hypothesis is.

(c) [6 pts] Give a recurrence for the running time $T(n)$ of the function. You may simplify constant factors.