Preliminary Examination I
Computer Science 312, Cornell University 8 March 2007

Before starting the exam, write your name on this page and your netid on both this page and the next page.

There are 4 problems on this exam. It is 7 pages long; make sure you have the whole exam. You will have 90 minutes in which to work on the problems. You will likely find some problems easier than others; read all problems before beginning to work, and use your time wisely. The prelim is worth 100 points total. The point breakdown for the parts of each problem is printed with the problem. Some of the problems have several parts, so make sure you do all of them!

This is an closed-book examination; you **may not** use outside materials, calculators, computers, etc.

Do all written work on the exam itself. If you are running low on space, write on the back of the exam sheets and be sure to write (OVER) on the front side. It is to your advantage to show your work—we will award partial credit for incorrect solutions that are headed in the right direction. If you feel rushed, try to write a brief statement that captures key ideas relevant to the solution of the problem.

If you finish in the last ten minutes of the exam, please remain in your seat until the end of the exam as a courtesy to your fellow students.

Name and NetID _____

Problem	Points	Score
1	14	
2	20	
3	24	
4	42	
Total	100	

1. True/False [14 pts]

   (parts a–g; 2 points off for each wrong answer, 1 point off for each blank answer)

a. ____ The SML `case` expression can bind new variables.

b. ____ Polymorphism allows a function to work on values of many different types.

c. ____ Anonymous functions in SML programs cannot be recursive.

d. ____ Glass-box test cases can be written using just the specification of the code that is being tested.

e. ____ If $f(n)$ is $O(g(n))$, then $g(n)$ is also necessarily $O(f(n))$.

f. ____ If function specification A has a weaker precondition than function B, then A may refine B.

g. ____ If an ADT implementation has no representation invariant, to be correct it must be able to handle any possible value of the declared representation type.

2. Substitution model [20 pts]

   For each of the following examples, give a correct type to fill in the box with. Then use the substitution model of evaluation to show the evaluation of the code. Show every step of evaluation. You may leave out type annotations when you show evaluation. You may use abbreviations as long as it is completely clear what is meant.

   (a) [6 pts]

```
let val x = (4, fn x:int => x+1)
    val (y:int, z: [    ]) = x
in
    z(y) + y
end
```

Type of z:

Evaluation:

```
let val x = (4, fn x => x+1)
    val (y, z) = x
in
    z(y) + y
end
```

$\longrightarrow$

```
let val (y, z) = (4, fn x => x+1)
in
    z(y) + y
end
```

$\longrightarrow$

```
(fn x => x+1)(4) + 4 ⟶ (4+1) + 4 ⟶ 5 + 4
```

(b) [6 pts]

```
let val r: [      ] = {name = "Bo", age = 24}
    fun name(s: string): char list =
        String.explode("x" ^ s)
in
   name (#name r)
end
```

(Hint: recall that `String.explode` *returns a list containing all the characters in a string)*

---

type of r:

(c) [8 pts]

```
let fun f(n: [    ]): int = case n of
    0 => 0
  | _ => n + f(n-1)
in
   f(1)
end
```

(Hint: define an abbreviation F that you can use to make writing out the evaluation much shorter)

---

type of n:

3. Zardoz [24 pts] (parts a–c)

For each box, provide a *value* that will cause the entire expression to evaluate to the correct answer: 42. (A list of values $[v_1, \ldots, v_n]$ is considered to be a value.)

(a) [8 pts]

```
let val lst: string list = [ ]
in
 case List.map (fn(x) =>
                    case Int.fromString(x) of
                        SOME n => n
                    | _ => 0)
               lst
 of
    x::y::z::_ => x*x + y*y + z*z
    | _ => 41
end
```

Your answer: [ ]

(b) [8 pts]

```
let val f = [ ]
    val r: Random.rand = Random.rand(1,1)
    val n = Random.randInt(r) mod 10
in
    f(n) + n
end
```

Your answer: [ ]

(**Hint:** Given a seed value of type `Random.rand`, `Random.randInt` returns a pseudo-random (i.e., unpredictable) integer.)

(c) [8 pts]

```
let val zaphod = [ ]
    val {a,b,c} = zaphod
in
    (a*10 + (List.length b)) div (case c of SOME x => 0 | _ => 2)
end
```

Your answer: [ ]

4. Data abstraction [42 pts] (parts a–g)

Consider the following signature describing a priority queue ADT:

```
signature PQUEUE = sig
  (* An 'a pqueue is a priority queue, in which elements are ordered,
   * and elements are removed in order.  Elements are of some type 'a.
   * The queue has an ordering function built in. Elements "equal" according
   * to this ordering have the same priority. *)
  type 'a pqueue
  (* create(l, ord) creates a priority queue containing all the
   * elements of l, ordered by ord.
   * Requires: ord is an ordering on 'a. *)
  val create: 'a list * ('a*'a->order) -> 'a pqueue
  (* first(q) is the lowest element in the queue according to its element
   * ordering. *)
  val first: 'a pqueue -> 'a
  (* push(q, x) is the priority queue containing all the elements of q,
   * and also x. *)
  val push: 'a pqueue * 'a -> 'a pqueue
  (* pop(q) is a pair (fst, q') where fst is the lowest element in q and
   * q' contains all the elements of q except fst. *)
  val pop: 'a pqueue -> 'a * 'a pqueue
end
```

Recall that the type `order` is a datatype defined as follows:

```
datatype order = LESS | EQUAL | GREATER
```

(a) [5 pts] Suggest an additional ADT operation that seems likely to be important to clients of this abstraction. Briefly justify adding this operation, and write the declaration and specification.

(b) [8 pts] The specifications of `first` and `pop` have some problems. Write better specifications.

Here is a possible implementation of the PQUEUE abstraction. Notice that the rep invariant does *not* require that the underlying list is kept in sorted order. The `first` function has not yet been implemented. (There is an extra copy of this code on the last page of the exam that you may remove for your reference.)

```
structure PQueue :> PQUEUE = struct
  type 'a pqueue = 'a list * ('a * 'a -> order)
  (* AF: (lst, ord) represents the priority queue containing all the elements
   *     in lst, ordered by the function ord.
   * RI: The first element of lst, if any, is less than or equal to
   *     all other elements in lst, according to ord. *)
  fun create(lst: 'a list, ord: 'a * 'a -> order) = case lst of
    [] => ([], ord)
  | h::t =>
        let val (min: 'a, rest:'a list) =
            foldl (fn (x: 'a, (curmin: 'a, elems:'a list)) =>
                     case ord(x, curmin) of
                       LESS => (x, curmin::elems)
                     | _ => (curmin, x::elems))
                  (h, [])
                  t
        in
          (min::rest, ord)
        end
  fun first(lst, ord) = raise Fail "Not implemented"
  fun push(q,x) = case q of
    ([], ord) => ([x], ord)
  | (y::t, ord) => (case ord(x,y) of
                      LESS => (x::y::t, ord)
                    | _ => (y::x::t, ord))
  fun pop(q) = let val (elems, ord) = q in
    (hd elems, create(tl elems, ord))
  end
end
```

(c) [5 pts] Suppose we make the call `create([15,10,33], Int.compare)` (where `Int.compare` has type `int*int->order`). The function `foldl` causes the anonymous function defined in `create` to be called twice. What are the successive values bound to the variables `x`, `curmin` and `elems` in those calls? What is the final result of the call to create?

Call 1: x = _____, curmin = _____, elems = _____

Call 2: x = _____, curmin = _____, elems = _____

Result = _____

(d) [5 pts] Explain in 2–3 sentences how it is possible that this implementation can work despite the fact that its representation of a priority queue does not keep the list in sorted order.

(e) [4 pts] Some of the implemented operations have constant asymptotic complexity, and some have linear complexity. Which is which?

(f) [5 pts] Give a very simple $O(1)$ implementation of `first`, and explain briefly why your implementation of `first` is correct with respect to the `PQUEUE` signature.

(g) [10 pts] Using the abstraction function and rep invariant, argue that the implementation of `push` is correct. Make sure you consider all the possible cases, and argue not only that the result is correct but that it satisfies any relevant invariants.

Extra copy of PQueue code: you may remove this

```
structure PQueue :> PQUEUE = struct
  type 'a pqueue = 'a list * ('a * 'a -> order)
  (* AF: (lst, ord) represents the priority queue containing all the elements
   *     in lst, ordered by the function ord.
   * RI: The first element of lst, if any, is less than or equal to
   *     all other elements in lst, according to ord. *)
  fun create(lst: 'a list, ord: 'a * 'a -> order) = case lst of
    [] => ([], ord)
  | h::t =>
        let val (min: 'a, rest:'a list) =
            foldl (fn (x: 'a, (curmin: 'a, elems:'a list)) =>
                     case ord(x, curmin) of
                       LESS => (x, curmin::elems)
                     | _ => (curmin, x::elems))
                  (h, [])
                  t
        in
          (min::rest, ord)
        end
  fun first(lst, ord) = raise Fail "Not implemented"
  fun push(x,q) = case q of
    ([], ord) => ([x], ord)
  | (y::t, ord) => (case ord(x,y) of
                      LESS => (x::y::t, ord)
                    | _ => (y::x::t, ord))
  fun pop(q) = let val (elems, ord) = q in
    (hd elems, create(tl elems, ord))
  end
end
```