CS312 Fall 2001
Prelim 1
October 18, 2001

Name____

Instructions

Write your name on the line above. There are seven questions on 10 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Ambiguous answers will be considered incorrect. The exam is closed book except for the handouts provided. Do not begin until instructed. You have 90 minutes. **Good luck!**

1	2	3	4	5	6	7	Σ
/14	/12	/7	/8	/7	/10	/9	/67

1. (14 points) Evaluate each of the following ML expressions and say what value will be produced. *Note*: One of the expressions is erroneous. For the erroneous expression, say approximately what error message will be produced.

   (a)
```
let
  val x = 4
  val x = x + 1
  val x = 1 - x
in
  x * x
end
```

   (b)
```
(fn x => fn y => y - x) 2 3
```

   (c)
```
let val x:int * int = 5 * 3
in ~x
end
```

   (d)
```
let
  fun shuffle (x:int list) (y:int list) : int list =
    case x of
      nil => y
    | u::z => u::(shuffle y z)
in
  shuffle [1,2,3] [4,5]
end
```

(e)
```
let
  val x = 3
  val y = 4
  val y =
    let val x = 5
    in x + y
    end
in
  x + y
end
```

(f)
```
foldl (op ^) "2" (map Int.toString [1,3])
```

(g)
```
let
  val imp = String.implode
  val chr = List.map Char.chr
  val inc = List.map (fn x => x + 1)
  val ord = List.map Char.ord
  val exp = String.explode
in
  (imp o chr o inc o ord o exp) "Mississippi"
end
```

2. (12 points) Fill in the box with a value that causes the entire expression to evaluate to 42. If impossible, say so and explain why. Example:

```
let
  val zardoz = [(42,43)]
in
  #1 zardoz
end
```

(a)
```
let
  val zardoz = [                    ]
in
  #zed zardoz
end
```

(b)
```
let
  fun zardoz (x:int list) (y:string list) : int =
    case (x,y) of
        (nil,_) => 0
      | (_,nil) => 1
      | (_::_::_,_) => 2
      | (_,_::_::_) => 3
      | _ => 42
in
  zardoz [                    ]
end
```

(c)
```
let
  val f = List.foldr (op +) 0
  val zardoz:int option list = [                    ]
  fun zed m =
    case m of NONE => 0
            | SOME i => i + 1
in
  f(map zed zardoz)
end
```

3. (7 points) Recall the definition of polymorphic binary trees given in class:

```
datatype 'a binary_tree =
  LEAF of 'a
| NODE of {left:'a binary_tree,
           right:'a binary_tree,
           data:'a}
```

Give a definition of `map` for binary trees that takes as input a function to apply to each data element and a binary tree and produces a new binary tree of the same shape with the function applied to each data element. Your `map` function should be curried and fully polymorphic.

```
fun map
```

4. (8 points) The following signature defines an interface for mutable (destructive) priority queues like the one discussed in class, which we showed how to implement using heaps:

```
signature MUTABLE_PRIOQ = sig
  type 'a prioq
  exception EmptyQueue
  val empty : ('a * 'a -> order) -> 'a prioq
  val insert : 'a prioq -> 'a -> unit
  val extractMin : 'a prioq -> 'a
end
```

The first argument to `empty` is a function that allows the priorities of elements to be compared. This version is called *mutable* or *destructive* since the `insert` and `extractMin` operations directly modify the priority queue that is passed to them as an argument. The function `extractMin` raises `EmptyQueue` if it is called with an empty queue.

One might also define a functional (non-destructive) version of priority queues with signature

```
signature FUNCTIONAL_PRIOQ = sig
  type 'a prioq
  exception EmptyQueue
  val empty : ('a * 'a -> order) -> 'a prioq
  val insert : 'a prioq -> 'a -> 'a prioq
  val extractMin : 'a prioq -> 'a * 'a prioq
end
```

Here the new priority queue after an `insert` or `extractMin` is returned as part of the output, and the original input priority queue is not modified.

Write a function `extract2nd` that is similar to `extractMin`, except that it retrieves the element with the second smallest priority instead of the smallest. The element with the smallest priority should remain in the queue. Do this for both (a) the mutable version, (b) the functional version. Implement your function in terms of functions already in the signature.

(enter answers on next page)

(a)

(b)

5. (7 points) Match each erroneous expression in (a)–(g) with the error message in (A)–(G) it would produce. Write the label of the error message in the box to the left of the corresponding expression.

	(a) `fun X f = f x`
	(b) `fun f(x:string):int = if size x > 312 then x else "xyzzy"`
	(c) `fun f(x:int):string = if x < 0 then ~1 else 1`
	(d) `(fn x => x + 312) "xyzzy"`
	(e) `(fn x => x ^ "xyzzy") 312`
	(f) `"xyzzy" ^ 312`
	(g) `if 1 < 2 then "xyzzy" else 312`

(A)
```
Error: operator and operand don't agree [literal]
  operator domain: string
  operand:         int
  in expression: ...
```

(B)
```
Error: operator and operand don't agree [literal]
  operator domain: int
  operand:         string
  in expression: ...
```

(C)
```
Error: operator and operand don't agree [literal]
  operator domain: string * string
  operand:         string * int
  in expression: ...
```

(D)
```
Error: unbound variable or constructor: x
```

(E)
```
Error: right-hand-side of clause doesn't agree with function
result type [literal]
  expression:   int
  result type:  string
  in declaration: ...
```

(F)
```
Error: right-hand-side of clause doesn't agree with function
result type [tycon mismatch]
  expression:   string
  result type:  int
  in declaration: ...
```

(G)
```
Error: types of rules don't agree [literal]
  earlier rule(s): bool -> string
  this rule:       bool -> int
  in rule: ...
```

6. (10 points) Match each declaration of `f` in (a)–(j) with the resulting type of `f` in (A)–(J). Write the label of the type in the box to the left of the corresponding declaration.

(a) `fun f x y = let val x = y in SOME (x*y) end`

(b) `fun f x y = let val x = y in SOME (x^y) end`

(c) `fun f x = case x of NONE => nil | SOME y => y`

(d) `fun f(x,y) = case x of SOME y => SOME (y+1) | NONE => NONE`

(e) `fun f x y = if x = y then SOME(x,y) else NONE`

(f) `fun f x y = if x then NONE else SOME (not x)`

(g) `fun f(x,y) = case x of NONE => nil | SOME z => [y + z]`

(h) `fun f y = let val f = 2*y in SOME (f*y) end;`

(i) `fun f x y = let val z = x*y in (SOME z,z) end`

(j) `fun f(x,y) = case y of nil => x | u::t => SOME u :: f(x,t)`

(A) `'a list option -> 'a list`

(B) `int option * 'a -> int option`

(C) `bool -> 'a -> bool option`

(D) `'a option list * 'a list -> 'a option list`

(E) `int -> int option`

(F) `int -> int -> int option * int`

(G) `''a -> ''a -> (''a * ''a) option`

(H) `int option * int -> int list`

(I) `'a -> int -> int option`

(J) `'a -> string -> string option`

7. (9 points) Consider the recurrence

$$\begin{aligned} T(0) &= 1 \\ T(n+1) &= T(n) + 4n + 3 \end{aligned}$$

Prove by induction on n that the solution to this recurrence is $T(n) = 2n^2 + n + 1$. Label clearly the basis and the induction step parts of your proof. In the induction step, state the induction hypothesis, and indicate exactly where in your argument it is used. Make sure you are perfectly clear in this—any ambiguity whatsoever will result in deduction of points.