# 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 | $\Sigma^a$ |
|---|---|---|---|---|---|---|---|
| /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
```
   $\boxed{\texttt{16}}$

   (b)
```
(fn x => fn y => y - x) 2 3
```
   $\boxed{\texttt{1}}$

   (c)
```
let
  val s:string = "xyzzy"
  val c:char = #"c"
  val x:int * int = String.size s * Char.ord c
in
  ~x
end
```
   $\boxed{\texttt{Type mismatch in declaration of x}}$

   (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
```
   $\boxed{\texttt{[1,4,2,5,3]}}$

(e)
```
let
  val x = 3
  val y = 4
  val y =
    let val x = 5
    in x + y
    end
in
  x + y
end
```
$\boxed{\texttt{12}}$

(f)
```
foldl (op ^) "2" (map Int.toString [1,3])
```
$\boxed{\texttt{"312"}}$

(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
```
$\boxed{\texttt{"Njttjttjqqj"}}$

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 = {zed=42}
in
  #zed zardoz
end
```

(b)
```
let
  fun zardoz (x:int list) (y:string list) : int =
    case (x,y) of
        (nil,_) => 38
      | (_,nil) => 39
      | (_::_::_,_) => 40
      | (_,_::_::_) => 41
      | _ => 42
in
  zardoz [312] ["312"]
end
```

(c)
```
let
  val f = List.foldr (op +) 0
  val zardoz:int option list = [SOME 41]
  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, similar to `List.map`.

```
fun map (f:'a -> 'b) (t:'a binary_tree) : 'b binary_tree =
  case t of
    LEAF x => LEAF (f x)
  | NODE {left,right,data} =>
     NODE {left=map f left, right=map f right, data=f data}
```

4. (8 points) Consider the following signature definitions.

```
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

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
```

These describe interfaces for mutable (destructive) and functional (non-destructive) priority queues, respectively. We showed in class how to implement mutable priority queues using heaps. The difference is that in the mutable version, the `insert` and `extractMin` operations modify the input priority queue as a side effect, whereas in the functional version, the new priority queue after an `insert` or `extractMin` is part of the return value, and the original input priority queue is not changed. In both versions, `EmptyQueue` is raised if `extractMin` is called with an empty queue.

Add a function `extract2nd` to both signatures. This function 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. In both cases, show how to implement `extract2nd` in terms of functions already in the signature.

(enter answers on next page)

(a)
```
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
  val extract2nd : 'a prioq -> 'a
end
```

```
fun extract2nd(p:'a prioq) : 'a = let
  val m1 = extractMin p
  val m2 = extractMin p
in
  insert p m1;
  m2
end
```

(b)
```
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
  val extract2nd : 'a prioq -> 'a * 'a prioq
end
```

```
fun extract2nd(p:'a prioq) : 'a * 'a prioq = let
  val (m1,p) = extractMin p
  val (m2,p) = extractMin p
  val p = insert p m1
in
  (m2,p)
end
```

5. (7 points) Match each erroneous expression (a)–(g) with its error message (A)–(G). Write the label of the error message in the box to the left of the corresponding expression. The correspondence is one-to-one.

$\boxed{\text{(D)}}$ (a) `fun X f = f x`

$\boxed{\text{(F)}}$ (b) `fun f(x:string):int = if size x > 312 then x else "xyzzy"`

$\boxed{\text{(E)}}$ (c) `fun f(x:int):string = if x < 0 then  1 else 1`

$\boxed{\text{(B)}}$ (d) `(fn x => x + 312) "xyzzy"`

$\boxed{\text{(A)}}$ (e) `(fn x => x ^ "xyzzy") 312`

$\boxed{\text{(C)}}$ (f) `"xyzzy" ^ 312`

$\boxed{\text{(G)}}$ (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 (a)–(j) with the resulting functional type (A)–(J) of `f`. Write the label of the type in the box to the left of the corresponding declaration. The correspondence is one-to-one.

$\boxed{\text{(I)}}$ (a) `fun f x y = let val x = y in SOME (x*y) end`

$\boxed{\text{(J)}}$ (b) `fun f x y = let val x = y in SOME (x^y) end`

$\boxed{\text{(A)}}$ (c) `fun f x = case x of NONE => nil | SOME y => y`

$\boxed{\text{(B)}}$ (d) `fun f(x,y) = case x of SOME y => SOME (y+1) | NONE => NONE`

$\boxed{\text{(G)}}$ (e) `fun f x y = if x = y then SOME(x,y) else NONE`

$\boxed{\text{(C)}}$ (f) `fun f x y = if x then NONE else SOME (not x)`

$\boxed{\text{(H)}}$ (g) `fun f(x,y) = case x of NONE => nil | SOME z => [y + z]`

$\boxed{\text{(E)}}$ (h) `fun f y = let val f = 2*y in SOME (f*y) end`

$\boxed{\text{(F)}}$ (i) `fun f x y = let val z = x*y in (SOME z,z) end`

$\boxed{\text{(D)}}$ (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.

Basis: For $n = 0$, we have

$$\begin{aligned} T(0) &= 1 && \text{by the definition of } T \\ &= 2 \cdot 0^2 + 0 + 1 && \text{by elementary algebra.} \end{aligned}$$

Induction step: The induction hypothesis is $T(n) = 2n^2 + n + 1$. Under this assumption, we wish to show that $T(n+1) = 2(n+1)^2 + (n+1) + 1$. We have

$$\begin{aligned} T(n+1) &= T(n) + 4n + 3 && \text{by the definition of } T \\ &= (2n^2 + n + 1) + 4n + 3 && \text{by the induction hypothesis} \\ &= 2(n+1)^2 + (n+1) + 1 && \text{by elementary algebra.} \end{aligned}$$