# MCS287 Final Exam

Serial #:

This exam is a closed-book, individual-work exam. It is also closed notes, with the exception of three $8.5 \times 11$ inch handwritten sheets of notes (both sides OK). Please write your name only on this page. Do not turn the page until instructed, in order that everyone may have the same time. Then, be sure to look at all problems before deciding which one to do first. Some problems are easier than others, so plan your time accordingly. You have 120 minutes to work.

Write the answer to each problem on the page on which that problem appears. You may use the back of that same sheet if you need additional space. You may also request additional paper, which should be labeled with your test number and the problem number.

Name: ________________________________

| Problem | Page | Possible | Score |
|---|---|---|---|
| 1 | 2 | 12 | |
| 2 | 3 | 12 | |
| 3 | 4 | 12 | |
| 4 | 5 | 12 | |
| 5 | 5 | 12 | |
| 6 | 5 | 12 | |
| 7 | 6 | 12 | |
| **Total** | | 84 | |

1. [ **12 Points** ] Use $\beta$-reduction (possibly repeatedly) to reduce each of the following to normal form:

```
((lambda (x)
   ((lambda (x)
      x)
    (x x)))
 y)
```

```
((lambda (x)
   (lambda (y)
     (y x)))
 y)
```

```
(((lambda (x)
    (lambda (y)
      x))
  w)
 z)
```

2. [ **12 Points** ] Write a Scheme expression equivalent to the following but without using `letrec` or `define`:

```
(letrec ((f (lambda (n)
              (if (= n 0)
                  1
                  (* (f (- n 1))
                     n)))))
  (f k))
```

3. [ **12 Points** ] Write a Ted expression that evaluates to 4 using call-by-reference and 20 using call-by-value-result, assuming that the direct array model is used in both cases. Draw diagrams to illustrate evaluation with each parameter passing mechanism. (You will normally need more than two diagrams for each mechanism: an initial and final snapshot generally will not suffice to show the process.)

4. [ **12 Points** ] Assume the original type checker from section 13.1 (the starting point of your lab) is being used. For each of the following Ted expressions, tell what type the checker would return (if the expression is well typed) or what error message the type checker would produce (if the expression is not well typed).

   (a) `(proc (x) 3)(5)`

   (b) `proc (x) if x then x else x`

   (c) `let x = true in if x then x else x`

   (d) `let f = assert (-> (int) int) : 3 in f(5)`

   (e) `let f = proc (x) 3 in +(f(5), f(true))`

   (f) `proc (f) if f(f(7)) then 3 else 4`

5. [ **12 Points** ] Do the same as problem 4 except with the monomorphic type inference system from section 13.3.

   (a) `(proc (x) 3)(5)`

   (b) `proc (x) if x then x else x`

   (c) `let x = true in if x then x else x`

   (d) `let f = assert (-> (int) int) : 3 in f(5)`

   (e) `let f = proc (x) 3 in +(f(5), f(true))`

   (f) `proc (f) if f(f(7)) then 3 else 4`

6. [ **12 Points** ] Do the same as problem 4 except with the polymorphic type inference system from section 13.4.

   (a) `(proc (x) 3)(5)`

   (b) `proc (x) if x then x else x`

   (c) `let x = true in if x then x else x`

   (d) `let f = assert (-> (int) int) : 3 in f(5)`

   (e) `let f = proc (x) 3 in +(f(5), f(true))`

   (f) `proc (f) if f(f(7)) then 3 else 4`

7. [ **12 Points** ] Suppose we apply `unify` to the types `(-> (bool, t1) t1)` and `(-> (t2, t2) int)`, where `t1` and `t2` are type variables (tvars). Will this succeed or fail? If it succeeds, specify the type variable bindings that result. If it fails, explain why by telling what variable bindings have already been done prior to the point where failure is detected, and what clash is then found.