Final Exam: COSE212 Programming Languages, Fall 2024

Student Id: Student Name:

Problem 1. (10pts) Define the following functions using

`fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`.

1. `map : ('a -> 'b) -> 'a list -> 'b list`

2. `filter : ('a -> bool) -> 'a list -> 'a list`

Problem 2. (10pts) Write a higher-order function

```
dropWhile : ('a -> bool) -> 'a list -> 'a list
```

that removes elements from the beginning of a list as long as they satisfy a given predicate. For instance,

```
dropWhile (fun x -> x mod 2 = 0) [2;4;7;4;9]
```

evaluates to `[7;4;9]` and

```
dropWhile (fun x-> x>5) [1;3;7]
```

is `[1;3;7]`.

```
let rec dropWhile p l =
match l with
| [] -> (1)
| hd::tl -> (2)
```

Complete (1) and (2).

Problem 3. (10pts) JavaScript programs often use closures: e.g.,

```
function create(init) {
  return function (step) {
    init += step;
    return init;
  };
}
var inc = create(5);
var a = inc(1);
var b = inc(2);
```

What are the values of `a` and `b` at the end of the program? Explain how these values are computed.

Hint: the JavaScript code can be translated to our language with implicit references as follows:

```
let create =
  proc (init)
    (proc (step) (init := init + step; init)) in
let inc = (create 5) in
let a = (inc 1) in
let b = (inc 2) in
(* What are the values of a and b here? *)
```

Problem 4. (10pts) Consider the program:

```
proc (maker)
  proc (x)
    if iszero (x) then 0
    else (((maker maker) (x-1)) + 4)
```

1. Does this program get accepted by the simple type system? Explain why.

2. Does this program get accepted by the let-polymorphic type system? Explain why.

Problem 5. (10pts) Consider the following program:

```
let f = proc (x) x in
  if (f (iszero 0)) then (f 11) else (f 22)
```

1. Is the program accepted by the simple type system? Explain why.

2. Is the program accepted by the let-polymorphic type system? Explain why.

Problem 6. (10pts) What is the value of the program?

```
let a = 5 in
  let y = 2 in
    let f = proc (x) (proc (y) (x+y+a)) in
      let x = 1 in
        let g = f y in
          let h = f y in
            (g 3) + (h 4)
```

1. With static scoping:

2. With dynamic scoping:

Problem 7. *(10pts) Evaluate the lambda terms:*

1. $(\lambda x.(\lambda x.x))\ y$

2. $(\lambda x.(\lambda y.x))\ y$

Problem 8. *(10pts) Define the translation to lambda calculus:*

1. $\underline{E_1 \text{ and } E_2} =$

2. $\underline{\text{succ } E} =$

Problem 9. (20pts) O/X questions. Leave a blank when uncertain. Each correct answer gets you 1 point but you lose 1 point for each wrong answer.

1. The following inductive definition

$$\frac{}{\text{leaf}} \qquad \frac{t_1 \quad t_2}{(n, t_1, t_2)}\ n \in \mathbb{Z}$$

defines the set of balanced binary trees. (A binary tree is balanced if the depth of the two subtrees of every node never differ by more than 1.)

2. The following program evaluates to 0.

```
let f = let cnt = ref 0
          in proc (x) (cnt := !cnt + 1; !cnt)
in let a = (f 0)
    in let b = (f 0)
        in (a - b)
```

3. With static scoping, the program

```
let a = 1 in
  let p = proc (b) (a+b) in
    let f = proc (a) (p a) in
      let a = 5 in
        (f 2)
```

evaluates to 3.

4. C supports call-by-reference parameter passing.
5. The following program evaluates to 12.

```
let f = proc (x) proc (y) (x := x + 1; x - y)
in ((f 44) 33)
```

6. The nameless representation of

```
proc (x) ((let x = 1 in x) + x)
```

is `proc ((let 1 in #0) + #0)`.

7. The following program is accepted by let-polymorphic type system.

```
let f = proc (x) x in (f f)
```

8. The static type system in OCaml accepts a program if and only if the program has no type errors at runtime.
9. $E_1 * E_2$ can be translated to lambda calculus by the rule $\underline{E_1 * E_2} = (\lambda m.\lambda n.\lambda s.\lambda z.m\ (n\ s)\ z)\ \underline{E_1}\ \underline{E_2}$
10. Every recursive function can be defined in terms of non-recursive functions.