First Exam
CS 1103 Computer Science I Honors

Fall 2017

Thursday September 28, 2017
Instructor Muller

KEY

Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please write your name **on the back** of this quiz.

This is a closed-notes and closed-book 35-minute exam. Computers, calculators, and books are prohibited. Work on this exam should stop at 9:35.

- Partial credit will be given so be sure to show your work.
- Feel free to write helper functions if you need them.
- **Please write neatly.**

Problem	Points	Out Of
1		1
2		1
3		2
4		2
5		1
6		3
Total		10

1. (1 Point) Show the step-by-step simplification of:

```
let n = 2 + 3 in n + n
```

Answer:

```
let n = 2 + 3 in n + n ->
  let n = 5 in n + n ->
  5 + 5 ->
  10
```

2. (1 Point) Show the step-by-step simplification of:

```
let f b = (not b) || b in match (f true) with | true -> 'A' | false -> 'B'
```

Answer:

```
let f b = (not b) || b in match (f true) with | true -> 'A' | false -> 'B' ->
match (not true) || true with | true -> 'A' | false -> 'B' ->
match false || true with | true -> 'A' | false -> 'B' ->
match true with | true -> 'A' | false -> 'B' ->
'A'
```

3. (2 Points) Consider the following three-dimensional shapes:

```
type shape = Cube of {width : float; height : float; depth : float}
           | Cylinder of {radius : float; height : float}
           | Sphere of float
```

Write a function `volume : shape -> float` that computes the volume of a 3D shape. FYI: The volume of a sphere of radius r is $(4/3)\pi r^3$.

Answer:

```
type shape = Cube of {width : float; height : float; depth : float}
           | Cylinder of {radius : float; height : float}
           | Sphere of float

(* volume : shape -> float
 *)
let volume shape =
  match shape with
  | Cube {width; height; depth} -> width *. height *. depth
  | Cylinder {radius; height} -> height *. 3.14 *. radius ** 2.0
  | Sphere radius -> (4.0 /. 3.0) *. 3.14 *. radius ** 3.0
```

4. (2 Points) Let `ns` be a list of integers in ascending order. Write a function

```
insert : int -> int list -> int list
```

such that a call `(insert n ns)` returns a list containing all of the elements of `ns` and also `n` and in which the numbers are in ascending order. For example, the call `(insert 3 [1; 2; 2; 3; 8])` would return the list `[1; 2; 2; 3; 3; 8]`.

Answer:

```
let rec insert n ns =
  match ns with
  | [] -> [n]
  | m :: ms -> (match m < n with
                | true  -> m :: insert n ms
                | false -> n :: ns)
```

5. (1 Point) How much work does `insert` do?

6. (3 Points) Write a function `topN : int -> int list -> int list` such that a call `(topN n ns)` returns the largest `n` integers in `ns`. For example, `(topN 3 [2; 4; 3; 4; 5; 1])` would return the list `[4; 4; 5]` because those are the largest 3 values in the list. You can assume that `n` is less than or equal to `(List.length ns)`. Note that the list may contain duplicates.

Answer:

```
(* largest : int list -> int * int list

   The call (largest ns) returns (m, ms) where is the largest
   value in ns and ms is ns - {m}.
*)
let rec largest ns =
  match ns with
  | [] -> failwith "empty list of numbers"
  | [n] -> (n, [])
  | n1 :: n2 :: ns ->
    let (min, max) = (min n1 n2, max n1 n2) in
    let (largest, others) = largest (max :: ns)
    in
    (largest, min :: others)

let topN n ns =
  let rec repeat n ns answer =
    match n = 0 with
    | true  -> answer
    | false -> let (largest, others) = largest ns
               in
               repeat (n - 1) others (largest :: answer)
  in
  repeat n ns []
```