First Quiz
CS 1103 Computer Science I Honors

Fall 2016

Thursday September 29, 2016
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 quiz. Computers, calculators, and books are prohibited. Work on this quiz should stop at 9:35.

Do problems 1, 2 and 3, then do EITHER problems 4 and 5 OR problem 6 for a total of 10 points.

- 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		3
5		3
6		6
Total		10

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

```
match (2 + 3) < 4 with | true -> "Boston" | false -> "College"
```

Answer:

```
match (2 + 3) < 4 with | true -> "Boston" | false -> "College" ->
match 5 < 4 with | true -> "Boston" | false -> "College" ->
match false with | true -> "Boston" | false -> "College" ->
"College"
```

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

```
match (let b = not false in false || b) with | true -> 2 + 3 | false -> 2 * 3
```

Answer:

```
match (let b = not false in false || b) with | true -> 2 + 3 | false -> 2 * 3 ->
match (let b = true in false || b) with | true -> 2 + 3 | false -> 2 * 3 ->
match (false || true) with | true -> 2 + 3 | false -> 2 * 3 ->
match true with | true -> 2 + 3 | false -> 2 * 3 ->
2 + 3 ->
5
```

3. (2 Points) Write a function `val monus : int -> int -> int` such that a call `(monus m n)` computes the difference of **m** and **n** but which returns **0** if the difference is negative. For example, the call `(monus 30 20)` should return **10** and the call `(monus 20 30)` should return **0**.

Answer:

```
(* val monus : int -> int -> int
 *)
let monus m n = max 0 (m - n)
```

4. (3 Points) Recall the type `fruit` as defined in class:

```
type fruit = Orange | Apple | Banana | Lemon
```

Write a function `citrusCount : fruit list -> int` such that a call `(citrusCount fruits)` returns the number of citrus fruits in the list. For example, the call

```
(citrusCount [Lemon; Orange; Apple; Lemon])
```

should return **3**.

Answer:

```
type fruit = Orange | Apple | Banana | Lemon

(* citrusCount : fruit list -> int
 *)
let rec citrusCount fruits =
  match fruits with
  | [] -> 0
  | Orange :: fruits | Lemon :: fruits -> 1 + citrusCount fruits
  | _ :: fruits -> citrusCount fruits
```

5. (3 Points) OCaml's notation for implementing the exponentiation operation m^n for floating point numbers m and n is `m ** n`. Write a function `power : int -> int -> int` such that a call `(power m n)` computes m^n for integers m and n. Recall that for any m, $m^0 = 1$. Do not use OCaml's `**` operator in your solution.

Answer:

```
let rec power m n =
  match n = 0 with
  | true  -> 1
  | false -> m * (power m (n - 1))
```

6. (6 Points) Write a function `histogram : float list -> Image.t list` such that a given call of the function `(histogram ns)` returns a list of horizontal bars (i.e., rectangles) representing the relative values in `ns`. Your `histogram` function is not responsible for displaying the rectangles, just creating them. For example, the call

   ```
   (histogram [1.; 2.; 5.; 3.; 2.])
   ```

   would return a list of rectangles that might be displayed as shown below. You may assume the existence of variables `displayWidth` and `displayHeight` as usual and you should arrange for the largest value in the list to occupy 100% of the width of the display. The width of the bars corresponding to the non-max values should be proportional.

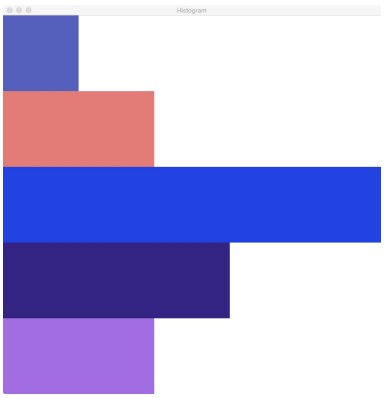

Figure 1: Displaying the list of bars resulting from a call (histogram [1.; 2.; 5.; 3.; 2.]).

Answer:

```
(* histogram : float list -> Image.t list
*)
let histogram ns =
  let max = maxList ns in
  let height = displayHeight /. (float (List.length ns)) in
  let rec repeat ns =
    match ns with
    | [] -> []
    | n :: ns ->
       let width = displayWidth *. (n /. max) in
       let color = Cs1103.randomColor() in
       let bar = Image.rectangle width height color
       in
       bar :: repeat ns
  in
  repeat ns
```