First Exam
CS 1101 Computer Science I
Fall 2015

KEY

Tuesday October 6, 2015
Instructor Muller
Boston College

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 exam.

This is a closed-notes and closed-book exam. Computers, calculators, and books are prohibited.

This is a 20 point exam. Answer all of questions 1.1, 1.2, 2.3, 2.5, 2.14 and 2.16. These total to 12 points. Choose any other problems totaling to **exactly 8** more points to arrive at 20. Circle the numbers of the problems that you want graded. If you select a set of problems totaling to more than 20 points, they'll be graded on scale of 20 points max and the lowest resulting problem scores will count toward the total.

- 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		**1**	**Required**
1.2		**1**	**Required**
2.1		1	
2.2		2	
2.3		**2**	**Required**
2.4		2	
2.5		**2**	**Required**
2.6		2	
2.7		2	
2.8		2	
2.9		2	
2.10		2	
2.11		2	
2.12		3	
2.13		3	
2.14		**3**	**Required**
2.15		3	
2.16		**3**	**Required**
2.17		3	
2.18		5	
Total		**20**	

Section 1

1. (1 Point) For each of the following, indicate what would happen if the code was evaluated in an OCaml REPL. If the code would produce an error, what error? If the code would produce a value, show all of the simplification steps.

   (a)
   ```
   let f x = x * 2;;
   let g y = y * (f y);;

   # g(4 + 1);;
   ```

   Answer:

   ```
   g(4 + 1) --> g(5)
            --> 5 * (f 5)
            --> 5 * (5 * 2)
            --> 5 * 10
            --> 50
   ```

   (b)
   ```
   let g x = x * z in
   let f z = (g 0, g z)
   in
   f 4;;
   ```

   Answer:

   ```
   Error: Unbound variable z.
   ```

2. (1 Point) For each of the following, indicate what would happen if the code was evaluated in an OCaml shell. If the code would produce a value, what value would it produce? If the code would produce an error, what error?

   (a)
   ```
   let f x y =
     let g x = x + y
     in
     g (x + y)
   in
   f 1 (g 2);;
   ```

   Answer:

   ```
   Error unbound variable g.
   ```

   (b)
   ```
   let f x y = x @ y in
   let g x z = x @ (f z z)
   in
   g ['A'] ['B']
   ```

   Answer:

   ```
   ['A'; 'B'; 'B']
   ```

Section 2

1. (1 Point) OCaml's *Pervasives* module has a two-argument function `max : a' -> 'a -> 'a` that returns the maximum of the two arguments. Write a three-argument version

   ```
   val max3 : 'a -> 'a -> 'a -> 'a
   ```

   For example, the call `(max3 4 8 2)` should evaluate to `8`.

   Answer:

   ```
   (* max3 : 'a -> 'a -> 'a -> 'a
    *)
   let max3 m n o = max m (max n o)
   ```

2. (2 Points) Write a function `maxList : 'a list -> 'a` such that a call `(maxList xs)` returns the largest element of `xs`. You may assume that `xs` is non-empty.

   Answer:

   ```
   (* maxList : 'a list -> 'a
    *)
   let rec maxList ns =
     match ns with
     | [] -> failwith "cannot happen"
     | [n] -> n
     | n::ns -> max n (maxList ns)

   let maxList ns = List.fold_left max (List.hd ns) (List.tl ns)
   ```

3. (2 Points REQUIRED) Write a function `count : 'a -> 'a list -> int` such that a call `(count x xs)` returns the number of occurrences of `x` in `xs`. For example, the call `(count 'A' ['A'; 'B'; 'A'])` should return **2**.

Answer:

```
(* count : 'a -> 'a list -> int
*)
let rec count x xs =
  match xs with
  | [] -> 0
  | y::ys -> (if x = y then 1 else 0) + (count x ys)
```

4. (2 Points) Write a function `power : int -> int -> int` such that a call `(power m n)` returns $\mathbf{m^n}$. For example, the call `(power 2 3)` should return **8**. Remember that any number raised to the 0 power is 1.

Answer:

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

5. (2 Points REQUIRED) Write a function `sumSquares : int list -> int` such that a call `(sumSquares ns)` returns the sum of the squares in `ns`. For example, the call `(sumSquares [2; 3; 10])` should return **113** because $\mathbf{4} + \mathbf{9} + \mathbf{100}$ sums to **113**.

Answer:

```
(* sumSquares : int list -> int
 *)
let rec sumSquares ns =
  match ns with
  | [] -> 0
  | n::ns -> (n * n) + (sumSquares ns)

let sumSquares ns =
  let square n = n * n
  in
  List.fold_left (+) 0 (List.map square ns)
```

6. (2 Points) Write a function `range : int -> int list` such that a call `(range n)` returns the list `[0; 1; ...; n-1]`.

Answer:

```
(* range : int -> int list
 *)
let range n =
  let rec repeat i =
    match i = n with
    | true  -> []
    | false -> i::repeat (i + 1)
  in
  repeat 0
```

7. (2 Points) Write a function `scanOver :'a -> 'a list -> 'a list` such that a call `(scanOver x xs)` returns a list consisting of the elements in `xs` to the right of the leftmost sequence of `x`s. For example, the call `(scanOver 1 [1; 1; 2; 1; 2; 3])` should evaluate to the list `[2; 1; 2; 3]` and the call `(scanOver 2 [1; 1; 2; 1; 2; 3])` should evaluate to the list `[1; 1; 2; 1; 2; 3]`.

Answer:

```
(* scanOver : 'a -> 'a list -> 'a list
 *)
let rec scanOver x xs =
  match xs with
  | [] -> []
  | y::ys -> match x = y with
             | true  -> scanOver x ys
             | false -> xs
```

8. (2 Points) Write a function `removeAll : 'a -> 'a list -> 'a list` such that a call `(removeAll x xs)` returns a list that is just like `xs` but all the occurrences of `x` have been removed. For example, the call `(removeAll 'A' ['A'; 'B'; 'C'; 'A'])` should evaluate to the list `['B'; 'C']`.

Answer:

```
(* removeAll : 'a -> 'a list -> 'a list
 *)
let rec removeAll x xs =
  match xs with
  | [] -> []
  | y::ys -> let ans = removeAll x ys
             in
             match x = y with
             | true  -> ans
             | false -> y::ans
```

9. (2 Points) Write a function `nth : int -> 'a list -> 'a` such that a call `(nth n xs)` returns the **n**th element of `xs`, where the first element of the list is at index 0 and so forth. If `n` is greater than the length of the list your function should raise `failwith`.

Answer:

```
(* nth : int -> 'a list -> 'a
 *)
let rec nth n xs =
  match (n, xs) with
  | (_, []) -> failwith "cannot retrieve nth element of empty list"
  | (0, x::_) -> x
  | (n, _::xs) -> nth (n - 1) xs
```

10. (2 Points) Write a function `removeNth : int -> 'a list -> 'a list` such that a call `(removeNth n xs)` returns the list that is just like `xs` except that the nth element is gone. If `n` is greater than the length of the list, raise a `failwith`.

Answer:

```
(* removeNth : int -> 'a list -> 'a list
 *)
let rec removeNth n xs =
  match (n, xs) with
  | (_, []) -> failwith "cannot drop nth element from empty list"
  | (0, _::xs) -> xs
  | (n, x::xs) -> x::removeNth (n - 1) xs
```

11. (2 Points) Write a function `powers : int -> int -> int list` such that a call `(powers m n)` returns the list consisting of all of the powers of `m` from 0 to `n`. For example, the call `(powers 2 8)` should return the list **[1; 2; 4; 8; 16; 32; 64; 128; 256]**.

Answer:

```
(* powers : int -> int -> int list
 *)
let rec powers m n =
  let rec repeat n acc =
    match n < 0 with
    | true  -> acc
    | false -> repeat (n - 1) ((power m n)::acc)
  in
  repeat n []

let powers m n = List.map (fun n -> power m n) (range n)
```

12. (3 Points) The *proper divisors* of an integer `n` are the integer divisors that are less than `n`. For example, the proper divisors of 6 are 1, 2 and 3 and the proper divisors of 8 are 1, 2 and 4. A number is said to be *perfect* if it is equal to the sum of its proper divisors. For example, both 6 and 28 are perfect. Write a function `isPerfect : int -> bool` such that a call `(isPerfect n)` returns `true` if and only if `n` is perfect.

Answer:

```
let properFactors n =
  let rec repeat c =
    match c <= n / 2 with
    | true  -> let answer = repeat (c + 1)
               in
               (match n mod c = 0 with
                | true  -> c::answer
                | false -> answer)
    | false -> []
  in
  1::repeat 2

(* isPerfect : int -> bool
 *)
let isPerfect n = List.fold_left (+) 0 (properFactors n) = n
```

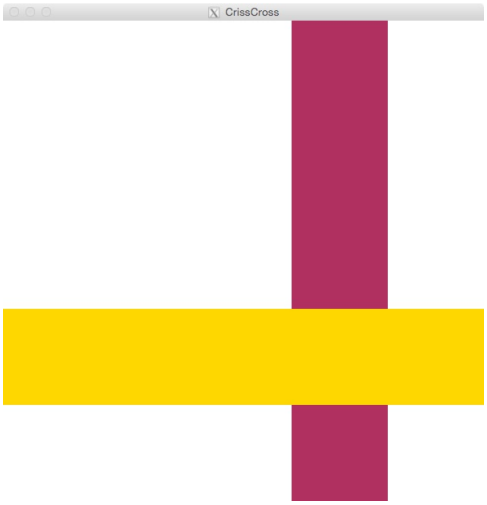
13. (3 Points) Write a function `crissCross : int -> int -> Image.t -> Image.t` such that a call `(crissCross m n image)` returns a new image with crossing gold and maroon stripes. The first argument specifies the grid size and the second specifies the row and column where the stripes are placed. For example, executing the code:

```
(* crissCross : int -> int -> Image.t -> Image.t
 *)
let crissCross m n image = YOUR CODE

let draw _ = crissCross 5 4 (Image.empty_scene displayWidth displayHeight)

let _ = World.big_bang ()
                       ~name:"CrissCross"
                       ~width:(Cs1101.f2I displayWidth)
                       ~height:(Cs1101.f2I displayHeight)
                       ~to_draw:draw
```

should return the image:



Answer:

```
(* crissCross : int -> int -> Image.t -> Image.t
 *)
let crissCross m n image =
  let size = displayWidth /. (float m) in
  let criss = Image.rectangle size displayHeight Color.maroon in
  let cross = Image.rectangle displayWidth size Color.gold in
  let xy = (float (n - 1)) *. size in
  let image1 = Image.place_image criss (xy, 0.) image
  in
  Image.place_image cross (0., xy) image1
```

14. (3 Points REQUIRED) An election is coming and the city of Indianapolis is hosting a debate. But there are too many candidates! Party elders have decided to invite only those candidates whose poll numbers are above average. Write a function `qualifiers : (string * int) list -> string list` such that a call `(qualifiers [(name1, poll1); ...; (namek, pollk)])` returns the list of candidates whose poll numbers are above average relative to the other candidates in the list.

Answer:

```
let qualifiers candidates =
  let n = List.length candidates in
  let polls = List.map snd candidates in
  let average = (List.fold_left (+) 0 polls) / n in
  let rec cull candidates =
    match candidates with
    | [] -> []
    | (name, p)::rest -> match p > average with
                         | true  -> name::cull rest
                         | false -> cull rest
  in
  cull candidates
```

15. (3 Points) Write a function `shuffle : int list -> int list` such that a call `(shuffle ns)` returns a list consisting of the integers in `ns` but in random order.

Answer:

```
(* shuffle : int list -> int list
 *)
let shuffle xs =
  let rec doShuffle xs acc =
    match xs with
    | [] -> acc
    | _  -> let n = Random.int (List.length xs) in
            let (pick, newList) = (nth n xs, dropNth n xs)
            in
            doShuffle newList (pick::acc)
  in
  doShuffle xs []
```

16. (3 Points REQUIRED) Many types of data such as audio or video contain long sequences of repeated values. For example, in audio, most of the sampled frequencies have 0 values for long stretches of time. In the *run length encoding* system, sequences of repeated values `v v ... v` are represented efficiently by *pairs* `(v, n)` where `n` is the length of the sequence. Write a function

```
runLengthEncoding : 'a list -> ('a * int) list
```

such that a call `(runLengthEncoding [x1; ...; xn])` returns a list of pairs `[(x1, n1); (x2, n2); ...]` where the input list `[x1; ...; xn]` starts out with a sequence of `n1` `x1`s, followed by `n2` `x2`s and so forth. For example, the call `(runLengthEncoding [0; 0; 0; 1; 1; 0; 0; 0; 0])` should return the list of pairs `[(0, 3); (1, 2); (0, 4)]`.

Answer:

```
let rec runLength x xs =
  match xs with
  | [] -> 1
  | y::ys -> (match x = y with
              | false -> 1
              | true  -> 1 + (runLength x ys))

(* runLengthEncoding : 'a list -> ('a * int) list
 *)
let rec runLengthEncoding items =
  match items with
  | [] -> []
  | item::items ->
     let itemCount = runLength item items in
     let followers = scanOver item items
     in
     (item, itemCount)::runLengthEncoding followers
```

17. (3 Points) Write a function `mostCommon : 'a list -> ('a * int)` such that a call

```
(mostCommon [x1; ...; xn])
```

returns a pair `(x, n)` where `x` is the most commonly occurring item in `xs` and `n` is the number of times it occurred. For example, the call `(mostCommon ['A'; 'B'; 'A'])` should return the pair `('A', 2)`. You may assume that the list is non-empty and you may return any winner in the case of ties.

Answer:

```
let rec pairMax pairs =
  match pairs with
  | [] -> failwith "pairMax: empty list has no max."
  | [(name, count)] -> (name, count)
  | (name1, count1)::(name2, count2)::rest ->
     match name1 > name2 with
     | true  -> pairMax ((name1, count1)::rest)
     | false  -> pairMax ((name2, count2)::rest)

(* mostCommon : 'a list -> ('a * int)
*)
let mostCommon xs = pairMax (List.map (fun x -> (x, count x xs)) xs)
```

18. (5 Points) Write a function `histogram : float list -> float -> Image.t` such that a call

```
(histogram [x1; ...; xn] max)
```

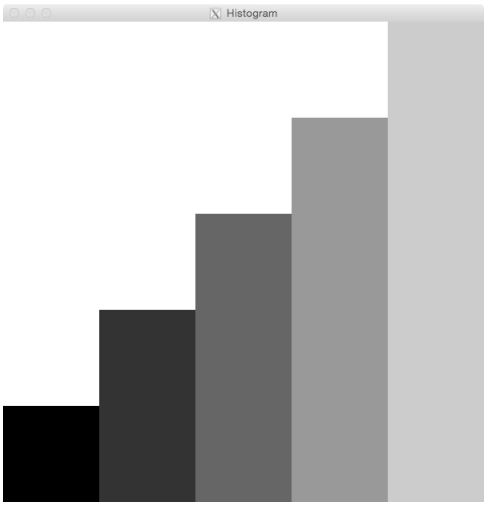
returns an `Image.t` with `n` equal-width vertical bars whose heights are proportional to the respective elements of the list. For example, executing the code:

```
(* histogram : float list -> float -> Image.t
 *)
let histogram data max = YOUR CODE

let draw _ = (histogram [1.; 2.; 3.; 4.; 5.] 5.0)

let _ = World.big_bang ()
                       ~name:"Histogram"
                       ~width:(Cs1101.f2I displayWidth)
                       ~height:(Cs1101.f2I displayHeight)
                       ~to_draw:draw
```

should produce the image:



Answer:

```
(* histogram : float list -> float -> Image.t
 *)
let histogram data max =
  let n = List.length data in
  let width = displayWidth /. (float n) in
  let shadeIncrement = 255 / n in
  let rec repeat i ns image =
    match ns with
    | [] -> image
    | m::ms -> let height = m /. max *. displayHeight in
               let shade = i * shadeIncrement in
               let color = Color.make_color shade shade shade in
               let bar = Image.rectangle width height color in
               let x = (float i) *. width in
               let y = displayHeight -. height in
               let newImage = Image.place_image bar (x, y) image
               in
               repeat (i + 1) ms newImage
  in
  repeat 0 data (Image.empty_scene displayWidth displayHeight)

let draw _ = histogram [1.; 2.; 3.; 4.; 5.] 5.0

let _ = World.big_bang ()
                       ~name:"Histogram"
                       ~width:(Cs1101.f2I displayWidth)
                       ~height:(Cs1101.f2I displayHeight)
                       ~to_draw:draw
```