First Exam
CS 101 Computer Science I
Section 03

KEY
Tuesday February 17, 2015
Instructor Muller
Boston College
Spring 2015

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.

Do any of the problems adding up to 7 points. In addition, do at least one of the 5 point problems for a total of 12 points. Circle the numbers of the problems that you want graded. If you select a set of problems totalling to more than 12 points, (e.g., 5 + 5 + 3), they will be graded on scale of 12 points max.

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

Problem	Points	Out Of
1		1
2		1
3		2
4		2
5		3
6		3
7		3
8		4
9		4
10		5 (one of 10 or 11)
11		5 (one of 10 or 11)
Total		12

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

   Answer:

   Boston

   (b)
   ```
   let f x y = (g x, g z)
   let g z = z * 2
   f 2 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);;
   f 2 3;;
   ```

   Answer:

   8

   (b)
   ```
   let f x y = g();;
   let g x y z = x + y;;
   f 1 6;;
   ```

   Answer:

   Error: wrong number of arguments to g.

3. (2 Points) Write a function `val bump : int -> int` that doubles even numbers and triples odd numbers. For example, the call `(bump 14)` should evaluate to `28` while the call `(bump 15)` should evaluate to `45`.

Answer:

```
(* val bump : int -> int
 *)
let bump n =
  match (n mod 2) == 0 with
  | true  -> n * 2
  | false -> n * 3
```

4. (2 Points) A customer is eligible for a discount if they are under 21 years of age and they have a gold card or if they are between the ages of 60 and 90. (Ninety one? You're out of luck.) Write a function `eligible : bool -> int -> bool` such that a call `(eligible goldCard age)` will return `true` if they are eligible for a discount and `false` otherwise.

Answer:

```
(* val eligible : bool -> int -> bool
 *)
let eligible goldCard age =
  (goldCard && age < 21) || (age >= 60 && age <= 90)
```

5. (3 Points) The built-in `mod` operator computes the integer remainder. In particular, the expression `m mod n` evaluates to the integer remainder when `m` is divided by `n`. Write `mod` as a function `mod : int -> int -> int` such that a call `(mod m n)` evaluates to the integer remainder when `m` is divided by `n`. Your solution should not use the built-in operator of the same name. For the purposes of this problem, you may assume that `m` and `n` are non-negative.

Answer:

```
(* val mod : int -> int -> int
*)
let rec mod m n =
  match m < n with
  | true  -> m
  | false -> mod (m - n) n
```

6. (3 Points) Write a function `swapAll : ('a * 'b) list -> ('b * 'a) list` such that a function call `(swapAll pairs)` returns a list of pairs that is like `pairs` but the components are swapped. For example, the call `(swapAll [(3, 4); (5, 6)])` should return the list of pairs `[(4, 3); (6, 5)]`.

Answer:

```
let rec swapAll pairs =
  match pairs with
  | []  -> []
  | (x, y)::rest -> (y, x)::swapAll rest
```

7. (3 Points) Write a function `cycles : int -> int -> int list` such that a call `(cycles m n)` generates a list of `n` cycles of the numbers 0 up to `m` - 1. For example, the call `(cycles 3 4)` should return the list `[0; 1; 2; 0; 1; 2; 0; 1; 2; 0; 1; 2]`. You may assume that `m` is a positive integer and `n` is non-negative.

Answer:

```
let rec upTo i m =
  match i = m with
  | true  -> []
  | false -> i::upTo (i + 1) m

(* val cycles : int -> int -> int list
*)
let rec cycles m n =
  match n = 0 with
  | true  -> []
  | false -> (upTo 0 m) @ (cycles m (n - 1))
```

8. (4 Points) Write an OCaml function `val rotateRight : int -> 'a list -> 'a list` such that a call `(rotateRight n xs)` rotates `xs` rightward `n` positions. By "rotate" we mean that elements that fall off the right, migrate to the left. For example, the call `(rotateRight 3 ['A'; 'B'; 'C'; 'D'])` should evaluate to the list `['B'; 'C'; 'D'; 'A']`. You may assume that `n` is non-negative.

Answer:

```
(* val rotateRight : int -> 'a list -> 'a list
*)
let rec rotateRight n xs =
  match n = 0 with
  | true  -> xs
  | false ->
      let (last, rest) = antiCons xs
      in
      rotateRight (n - 1) last::rest
```

9. (4 Points) Write an OCaml function `val split : int -> int list -> (int list * int list)` such that a call `(split n ns)` splits the list `ns` into a pair of list `(xs, ys)` where `xs` are all the numbers in `ns` that are less than or equal to `n` and `ys` are all the numbers in `ns` that are greater than `n`. For example, the call `(split 5 [4; 8; 3; 6; 5])` might return the pair `([4; 3; 5], [8; 6])`.

Answer:

```
(* val split : int -> int list -> (int list * int list)
*)
let rec split n ns =
  match ns with
  | []  -> ([], [])
  | m::ms ->
      let (leq, gtr) = split n ms
      in
      match m <= n with
      | true  -> (m::leq, gtr)
      | false -> (leq, m::gtr)
```

10. (5 Points) An election resulted in a tally of the votes represented in OCaml as a list of pairs:

```
let votes = [("Joe", 200); ("Mary", 300); ("Bob", 50)];;
```

Write a function `val winner : (string * int) list -> string` that will return the name of the winner of the election. For example, the call `(winner votes)` should evaluate to `"Mary"`. You may assume that the `votes` list is non-empty and that the election will not result in a tie.

Answer:

```
(* val winner : (string * int) list -> string
*)
let rec winner votes =
  match votes with
  | [] -> failwith "winner: cannot happen."
  | (name, _)::[] -> name
  | (name1, votes1)::(name2, votes2)::rest ->
        match votes1 > votes2 with
         | true  -> winner ((name1, votes1)::rest)
         | false -> winner ((name2, votes2)::rest);;
```

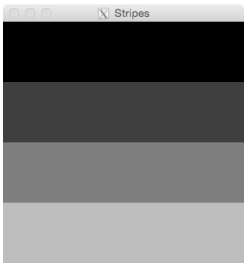
11. (5 Points) The `Color` module's `make_color` function has the type

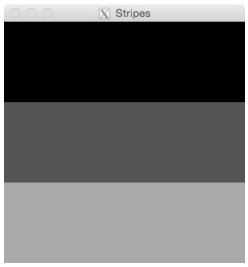
```
Color.make_color : int -> int -> int -> Color.t
```

In a call `(make_color r g b)`, the arguments `r`, `g` and `b` are expected to be integers in the range 0 to 255 specifying the amount (resp.) of *red*, *green* and *blue* in the returned color. When `r`, `g` and `b` are all the same, the returned color turns out to be a shade of *gray*. In particular, the call `(make_color 0 0 0)` makes *black*, `(make_color 255 255 255)` makes *white* and the call `(make_color 127 127 127)` makes a middling shade of *gray*.

Write a function `grayStripes : int -> unit` that when called as shown in the `draw` function below will render `n` horizontal stripes with successively lighter shades of gray. The top stripe should be pure *black* and successive stripes should be lighter shades, stopping short of *white*. For example, the call `(grayStripes 4)` as shown below would produce the image on the left and the call `(grayStripes 3)` would produce the image in the right.

```
open Color, Image, World

let (displayWidth, displayHeight) = (300, 300)

let grayStripes n = ...

let draw _ = grayStripes 4

let _ = big_bang () ~name:"Stripes" ~to_draw:draw
                    ~width:displayWidth ~height:displayHeight
```





Answer:

```
let grayStripes n =
  let stripeHeight = displayHeight / n in
  let shadeAmount = 255 / n in
  let rec repeat n image =
    match n = 0 with
    | true  -> image
    | false ->
       let y = stripeHeight * (n - 1) in
       let rgb = shadeAmount * (n - 1) in
       let shade = make_color rgb rgb rgb in
       let stripe = rectangle displayWidth stripeHeight shade in
       let newImage = place_image stripe (0, y) image
       in
       repeat (n - 1) newImage
  in
  repeat n (empty_scene displayWidth displayHeight)
```

Use this page for problem 11.