First Exam
CS 1101 Computer Science I

Section 03, Spring 2016

Tuesday March 1, 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 exam.

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

- 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		3	(only two of 5, 6 and 7)
6		3	
7		3	
8		4	(only two of 8, 9 and 10)
9		4	
10		4	
Total		20	

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 g z = z * 2 in
   let f x y = (g x, g z)
   in
   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)
   in
   f 3 4
   ```

   Answer:

   11

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

   Answer:

   Error: wrong number of arguments to g.

3. (2 Points) Write a function `val bump : int -> int` that doubles odd numbers and triples even numbers. For example, the call `(bump 14)` should evaluate to `42` while the call `(bump 15)` should evaluate to `30`.

Answer:

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

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)` returns `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)
```

3 Point Problems – choose only two

Circle the numbers of the two problems that you want graded.

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. Hint: consider repeated subtraction.

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 `snds : ('a * 'b) list -> 'b list` such that a function call `(snds pairs)` returns a list of the second components of the pairs. For example, the call `(snds [(1, 'A'); (2, 'B')])` should return the list `['A'; 'B']`. The built-in function `snd : 'a * 'b -> 'b` isn't needed for this solution but feel free to use it if you want to.

Answer:

```
let rec snds pairs =
  match pairs with
  | [] -> []
  | (_, snd) :: pairs' -> snd :: (snds pairs')

let snds pairs = List.map (fun (_, snd) -> snd) pairs

let snds pairs = List.map snd pairs
```

7. (3 Points) Write an OCaml function `val rotateLeft : int -> 'a list -> 'a list` such that a call `(rotateLeft n xs)` rotates `xs` leftward `n` positions. By "rotate" we mean that elements that fall off the left end, migrate to the right end. For example, the call `(rotateLeft 3 ['A'; 'B'; 'C'; 'D'])` should evaluate to the list `['D'; 'A'; 'B'; 'C']`. You may assume that `n` is non-negative.

Answer:

```
(* val rotateLeft : int -> 'a list -> 'a list
 *)
let rec rotateLeft n xs =
  match (n = 0, xs) with
  | (true, _)  -> xs
  | (false, x :: xs) -> rotateLeft (n - 1) (xs @ [x])
```

4 Point Problems – choose only two

Circle the numbers of the two problems that you want graded.

8. (4 Points) It's Super Tuesday and time to cull the field for the next debate. The debate organizers have analyzed the polls and gathered summary data in a list of pairs:

```
let candidates = [("Trump", 0.30); ("Rubio", 0.20); ("Cruz", 0.18); ("Carson", 0.02)]
```

The debate organizers wish to invite only candidates with above average poll numbers. Write a function

```
cull : (string * float) list -> (string * float) list
```

such that a call `(cull candidates)` returns the list of those candidates with above average poll numbers. For example, on the data above, since the average is **0.175**, the **cull** function would return the list of three candidates `[("Trump", 0.30); ("Rubio", 0.20); ("Cruz", 0.18)]`.

Answer:

```
(* val cull : (string * float) list -> (string * float) list
 *)
let cull candidates =
  let ave = average candidates
  in
  List.filter (fun (_, poll) -> poll > ave) candidates

let rec addUp ns =
  match ns with
  | [] -> 0.0
  | n :: ns -> n +. addUp ns

let average candidates =
  let numbers = List.map snd candidates in
  let total = addUp numbers
  in
  total /. (float (List.length numbers))

(* No need for addUp ...
 *)
let average candidates =
  let numbers = List.map snd candidates in
  let total = List.fold_left (+.) 0.0 numbers
  in
  total /. (float (List.length numbers))
```

9. (4 Points) We take our positional numeral system for granted but it was at least a couple of millennia in the making. The "positions" represent powers of 10 ascending from right to left: ones, tens, hundreds and so forth. For example, the decimal numeral for the present year, 2016, can be understood as

$$\begin{aligned} 2016_{10} &= 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 6 \times 10^0 \\ &= 2 \times 1000 + 0 \times 100 + 1 \times 10 + 6 \times 1 \\ &= 2000 + 0 + 10 + 6 \\ &= 2016 \end{aligned}$$

The system is so robust and flexible that it works for any base. For example, a numeral in base 3 can be understood in the same way:

$$\begin{aligned} 201_3 &= 2 \times 3^2 + 0 \times 3^1 + 1 \times 3^0 \\ &= 2 \times 9 + 0 \times 3 + 1 \times 1 \\ &= 18 + 0 + 1 \\ &= 19_{10} \end{aligned}$$

Write a function `toDecimal : int list -> int -> int` such that a call `(toDecimal digits base)` returns the decimal value of the list of digits. For example, the call `(toDecimal [2; 0; 1] 3)` should return `19`. Feel free to use the `List.rev` function which reverses a list and feel free to use the `**` operator on integers (rather than floats).

Answer:

```
let ipow m n = (int_of_float ((float m) ** (float n)))

(* toDecimal : int list -> int -> int
 *)
let toDecimal digits base =
  let rec repeat digits power acc =
    match digits with
    | [] -> acc
    | digit :: digits -> let n = digit * (ipow base power)
                         in
                         repeat digits (power + 1) (n + acc)
  in
  repeat (List.rev digits) 0 0
```

10. (4 Points) Given a decimal numeral X and base b, how can the numeral X_{10} be converted to the equivalent numeral in base b? How might we write a function `decimalTo : int -> int -> int list` so that a call such as `(decimalTo 19 3)` would evaluate to the list of digits `[2; 0; 1]` making up the base 3 numeral 201_3?

    Let's say we have a helper function:

    ```
    let div m n = (m / n, m mod n)
    ```

    which computes both the integer quotient and the remainder when m is divided by n. Then we can proceed in a repetitive process as follows:

    $$\begin{aligned} (\texttt{div } 19\ 3) &= (6,1) \\ (\texttt{div } 6\ 3) &= (2,0) \\ (\texttt{div } 2\ 3) &= (0,2) \end{aligned}$$

    In each iteration from one line to the next, the quotient of the previous step moves down to the left and the successive remainders make up the digits of the answer from right to left. When the quotient is zero the process is complete. Write the function `decimalTo`.

    Answer:

```
(* decimalTo : int -> int -> int list
 *)
let decimalTo n base =
  let rec repeat n acc =
    match n = 0 with
    | true  -> acc
    | false -> let (q, r) = div n base
               in
               repeat q (r :: acc)
  in
  repeat n []
```