Second Exam
CS 1101 Computer Science I
Fall 2015

KEY

Tuesday November 12, 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 12 point exam. Answer all questions in Section 1 and question 2.1. Answer either 2.2 or 2.3 but not both. Circle the number of the problem that you want graded.

- 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		**5**	**Required**
2.1		**3**	**Required**
2.2		4	
2.3		4	
Total		**12**	

Section 1 (5 Points Total)

1. (1 Point) In a sentence, what is the significance of the number 256 in computing?

   Answer: $2^8 = 256$**, there are 256 8-bit patterns.**

2. (1 Point) Is the following well-defined? If so, what is its value?

   ```
   let a = let b = a in 2 * b in a * a
   ```

   Answer: No, the occurrence of `a` initializing `b` is undefined.

3. (1 Point) Solve for X.

   (a) $86_{10} = X_{16}$.

   Answer: $X = 56$**.**

   (b) $11011_2 = X_8$.

   Answer: $X = 33$**.**

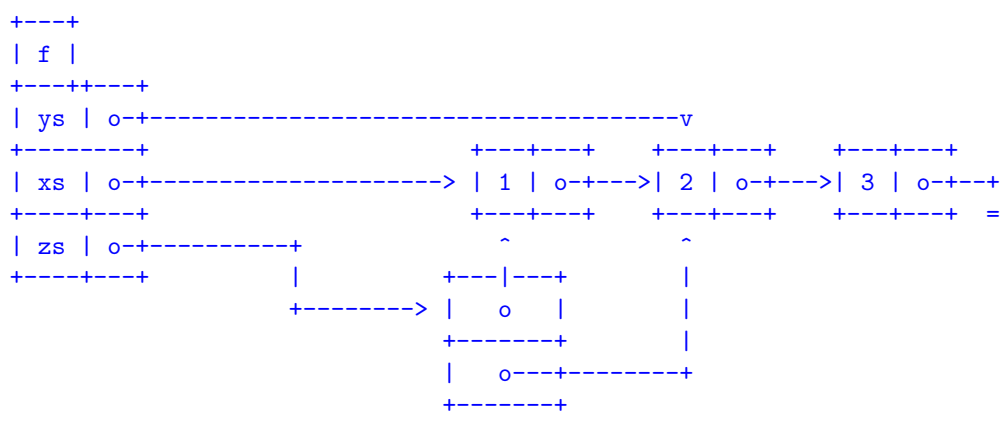
4. (2 Points) The `List.tl : 'a list -> 'a list` function returns the tail of a list. For example, the call `(List.tl [1; 2; 3])` evaluates to the list `[2; 3]`. Given the call, `(f [1; 2; 3])`, show the state of the stack and heap after (1) has executed but before (2) has executed.

```
let f xs =
  let ys = List.tl xs in
  let zs = (xs, ys)              (1)
  in
  zs                             (2)

(f [1; 2; 3])
```

Stack Heap

Answer:



Section 2 (7 Points Total)

Do problem 1 and **either 2 or 3 but not both**. Circle the number of the problem that you want graded.

1. (3 Points) Write a function `union : 'a list -> 'a list -> 'a list` such that a call `(union xs ys)` returns a list consisting of all elements in either `xs` or `ys`. The resulting list should have no duplicates. For example, the call `(union [3; 2; 1] [3; 2; 4])` should evaluate to the list `[3; 1; 2; 4]`.

   Answer:

```
let rec member x xs =
  match xs with
  | [] -> false
  | y::ys -> (x = y) || (member x ys)

let rec union xs ys =
  match xs with
  | []    -> ys
  | x::xs -> let answer = union xs ys
             in
             match (member x answer) with
             | true  -> answer
             | false -> x::answer
```

2. (4 Points) Write a function `product : 'a list -> 'b list -> ('a * 'b) list` such that a given call `(product xs ys)` returns a list of all pairs formed from elements of `xs` and `ys`. For example, the call `(product [1; 2] ['A'; 'B'])` should evaluate to `[(1, 'A'); (1, 'B'); (2, 'A'); (2, 'B')]`.

   Answer:

```
let rec product xs ys =
  let nested = List.map (fun x -> List.map (fun y -> (x, y)) ys) xs
  in
  List.fold_left (@) [] nested
```

3. (4 Points) Write a function `merge : int array -> int array -> int array` such that a given call `(merge a b)`, where `a` and `b` are arrays of integers in ascending order, returns an integer array containing the values in `a` and `b` in ascending order. For example, the call `(merge [|1; 3; 5|] [|2; 4|])` should return the 5-element array `[|1; 2; 3; 4; 5|]`.

Answer:

```
let merge a b =
  let na = Array.length a in
  let nb = Array.length b in
  let c = Array.make (na + nb) 0 in
  let rec repeat ai bi =
    match (ai < na, bi < nb) with
    | (true, true) -> if a.(ai) < b.(bi) then
                        begin
                          c.(ai+bi) <- a.(ai);
                          repeat (ai+1) bi
                        end
                      else
                        begin
                          c.(ai+bi) <- b.(bi);
                          repeat ai (bi+1)
                        end
    | (true, false) ->  c.(ai+bi) <- a.(ai);
                        repeat (ai+1) bi
    | (false, true) ->  c.(ai+bi) <- b.(bi);
                        repeat ai (bi+1)
    | (false, false) -> ()
  in
  repeat 0 0;
  c
```

Or, cutting corners:

```
let merge a b =
  Array.of_list (List.merge compare (Array.to_list a) (Array.to_list b))
```