CMSC330 Spring 2015 Midterm #2
9:30am/11:00am/12:30pm

Name: __________________________ UID: _______________________

Discussion Time (circle one): 10am  11am  12pm  1pm  2pm  3pm  
Discussion TA (circle one): Amelia  Casey  Chris  Mike  Elizabeth  Eric  Tommy

Instructions
● The exam has 3 pages; make sure you have them all.
● Do not start this test until you are told to do so!
● You have 75 minutes to take this midterm.
● This exam has a total of 100 points, so allocate 45 seconds for each point.
● This is a closed book exam. No notes or other aids are allowed.
● Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
● For partial credit, show all of your work and clearly indicate your answers.
● Write neatly. Credit cannot be given for illegible answers.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 OCaml typing</td>
<td>/19</td>
</tr>
<tr>
<td>2 OCaml execution</td>
<td>/12</td>
</tr>
<tr>
<td>3 OCaml coding</td>
<td>/20</td>
</tr>
<tr>
<td>4 Short answer, true/false</td>
<td>/18</td>
</tr>
<tr>
<td>5 Multithreading</td>
<td>/10</td>
</tr>
<tr>
<td>6 Lambda calculus</td>
<td>/12</td>
</tr>
<tr>
<td>7 Context Free Grammars</td>
<td>/9</td>
</tr>
<tr>
<td>Total</td>
<td>/100</td>
</tr>
</tbody>
</table>
1. **OCaml typing (19 pts)**

   a. (3 pts) What is the type of the following OCaml expression?
   
   ```
   [[1]; [2; 3]]
   ```
   
   **Answer:** int list list

   b. (3 pts) What is the type of \( f \) in the following definition?
   
   ```
   let f seed =
   let r = ref seed in
   (fun x -> r := !r + x; !r);
   ```
   
   **Answer:** int -> int -> int

   c. (4 pts) What is the type of \( \text{foo} \) in the following definition?
   
   ```
   let rec foo (x,y,l) =
   match l with
   | [] -> []
   | (h::t) ->
     if (h = x) then (y::(foo (x,y,t)))
     else (h::(foo (x,y,t)));
   ```
   
   **Answer:** 'a * 'a * 'a list -> 'a list

   d. (3 pts) Write an OCaml expression having type int -> int
   
   **Answer (sample):** fun x -> x + 1

   e. (3 pts) Write an OCaml expression having type 'a -> 'a list
   
   **Answer (sample):** fun x -> [x]

   f. (3 pts) Write an OCaml expression having type bool * int -> int
   
   **Answer (sample):** fun (x,y) -> if x then y else 1
2. **OCaml execution (12 pts):** What is the value of the variable result after executing the following code? If an error will occur, say what the problem is. (Note there are no syntax errors in these programs.)

   a. (3 pts)
   let g x =
      match x with
      h::t -> h;;
   let result = g [[9; 17]];;
   **Answer:** [9;17]

   b. (3 pts)
   let result = (fun x y -> x::y) (3 [4]);;
   **Answer:** ERROR, attempt to apply 3 as a function

   c. (3 pts)
   let rec foo (x,y,l) = match l with
      [] -> []
      | (h::t) ->
         if (h = x) then (y::(foo (x,y,t)))
         else (h::(foo (x,y,t)));;
   let result = foo("y","p",foo("p","c",[“p”;“l”;“a”;“y”]));;
   **Answer:** [“c”;”l”;”a”;”p”]

   d. (3 pts)
   let f seed =
      let r = ref seed in
      (fun x -> r := !r + x; !r)
   ;;
   let g = f 17;;
   let result = (g 1, g 2, g 3);;
   **Answer:** (23,22,20)
   
   Partial credit (2 pts): (18,20,23)
3. OCaml coding (20 pts). Complete two of the following three questions. If you do all three, all three will be graded, and the total will be scaled by ⅔ to be at most 20. You may write helper functions if you like. You may also use standard library functions, e.g., from the List and Pervasives modules, unless otherwise noted.

a. (10 pts) Write a function `multiply`, having type `int -> int -> int`, that returns the product of its two arguments. You may assume that both arguments are nonnegative, and your implementation may use the built-in + and - operators, but not any other built-in arithmetic operators.

Answer:

```ocaml
let rec multiply x y =
  if x = 0 then 0
  else y + multiply (x-1) y
```

b. (10 pts) Write a function `get_val` having type `‘a list -> int -> ‘a option` such that `get_val x n` returns `(Some z)` if z is the element of list x at index n (indexes start at 0) and returns None otherwise (i.e., if the list has at fewer than n+1 elements). Recall that

```
type ‘a option = Some of ‘a | None
```

Examples:

```
get_val [“hello”; “there”; “friend”] 0 ⇒ Some(“hello”)
get_val [1; 2; 3] 1 ⇒ Some(2)
get_val [1] 2 ⇒ None
```

Answer:

```ocaml
let rec get_val l i =
  if i<0 then None else
  match l with
  | [] -> None
  | h::t -> if i = 0 then (Some h) else get_val t (i-1)
```
c. (10 pts) Given the type `g_tree` defined as follows:

```ml
type g_tree =
  GNode of g_tree list
 | Leaf
```

Write a function `count` of type `g_tree -> int` that counts the number of leaves. It must take linear time (traverse the tree once). For example

```ml
count GNode([GNode([Leaf]);Leaf;Leaf]) => 3
count GNode([]) => 0
```

**Answer:**

```ml
let rec count t =
  match t with
  | Leaf -> 1
  | GNode(l) ->
    let rec aux gs =
      (match gs with [] -> 0 | h::t -> (count h) + (aux t)) in
    aux l
```

or

```ml
let rec count t =
  match t with
  | Leaf -> 1
  | GNode(l) ->
    List.fold_right (fun t a -> (count t) + a) l 0
```
4. Short answer, true/false (18 points).

a. (3 pts) When using closures to encode objects, where are an object’s private fields stored, so that they are private?
   Answer: In the closure’s environment

b. (3 pts) What is the relationship between a wait set and a condition variable?
   Answer: A wait set keeps track of the threads waiting on a condition variable

c. (3 pts) True or false: In a type-safe language, all type-correct programs are well-defined.
   Answer: true

d. (3 pts) True or false: static type checking occurs once the program starts to execute.
   Answer: false

e. (3 pts) True or false: reentrant locks are important for avoiding inadvertent self-deadlock.
   Answer: true

f. (3 pts) True or false: if OCaml had dynamic scoping, it would give the same answer as static scoping on the following program:

   \[
   \begin{align*}
   \text{let } f &\ x = x+1;; \\
   \text{let } g &\ y = (f\ y) + 1;; \\
   \text{let } f &\ x = x-1;; \\
   g &\ 2;;
   \end{align*}
   \]

   Answer: false
5. **Multithreading (10 points).**
   
a. (6 points) Given the following Ruby code, give two possible legal schedules in which the final value of $\texttt{var}$ is different; for each schedule also give the final value for $\texttt{var}$. A schedule is a sequence of events indicating the perceived order in which threads execute particular lines of code. For example, the schedule T1:1, T1:2, T2:1, T2:2 says that thread T1 executes line 1, then T1 executes line 2, then T2 executes line 1, and finally T2 executes line 2.

   \[ \texttt{var} = 1 \]

   ```ruby
   def driver_1()
     local_x = $\texttt{var}$
     $\texttt{var} = \texttt{local\_x + 1}$
   end
   
   def driver_2()
     local_y = $\texttt{var}$
     $\texttt{var} = \texttt{local\_y + 2}$
   end
   
   t1 = Thread.new{ driver_1() }
   t2 = Thread.new{ driver_2() }
   
   Answer(s):
   T1:1, T1:2, T2:3, T2:4 $\texttt{var}=4$
   T1:1, T2:3, T2:4, T1:2 $\texttt{var}=2$
   T1:1, T2:3, T1:2, T2:4 $\texttt{var}=3$
   *(there are others)*
   ```

b. (4 pts) Mark up the code above to show how you could use synchronization to ensure that only one final value of $\texttt{var}$ is possible.

   **Answer:** Create a monitor, put contents of driver_1() and driver_2() in synchronize block.
6. **Lambda calculus (12 points).** Evaluate the following lambda terms as much as possible. For full credit, show each beta reduction and alpha conversion you perform.

   a. (3 pts) \((\lambda x. x) z\)
   *Answer:* \((\lambda x. x) z \rightarrow z\)

   b. (3 pts) \((\lambda x. \lambda y. x) y z\)
   *Answer:* \((\lambda x. \lambda y. x) y z \rightarrow (\lambda y. z) y \rightarrow z y\)

   c. (3 pts) \((\lambda z. \lambda y. z) y x\)
   *Answer:* \((\lambda z. \lambda y. z) y x \rightarrow (\lambda z. \lambda w. z w) y x \rightarrow (\lambda w. y w) x \rightarrow y x\)

   d. (3 pts) Recall the encoding of Church numerals:
   
   \[
   \begin{align*}
   1 &= \lambda f. \lambda y. f y \\
   2 &= \lambda f. \lambda y. f(f y) \\
   3 &= \lambda f. \lambda y. f(f(f y)) \\
   (+ M N) &= \lambda x. \lambda z. ((M x)((N x) z))
   \end{align*}
   \]

   The following applies beta reductions to \((+ 1 2)\) and seems to end up with the term 2, instead of 3. There is a problem in one of the beta reductions (the grayed part); circle the problem and write a note to explain what’s wrong.

   
   \[
   \begin{align*}
   (+ 1 2) &= \lambda x. \lambda z. ((1 x)((2 x) z)) \\
   &= \lambda x. \lambda z. (((\lambda f. \lambda y. f(f y)) x)((2 x) z)) \\
   &\rightarrow \lambda x. \lambda z. (((\lambda f. \lambda y. f(f y)) x)((2 x) z)) \quad \text{\(\beta\)} \\
   &\rightarrow \lambda x. \lambda z. (((\lambda y. x(x y)) z) \quad \text{\(\alpha\)} \\
   &\rightarrow \lambda x. \lambda z. (x(x z)) \\
   &\rightarrow \lambda x. \lambda z. (x(x z)) \\
   &\rightarrow \lambda f. \lambda y. (f(f y)) \quad \text{Unsubstitute encoding for 2} \\
   &= 2
   \end{align*}
   \]
Context free grammars (9 pts). Consider the following grammar (where uppercase letters are non-terminals, lowercase letters and symbols are terminals, and $S$ is the start symbol).

$$
S \rightarrow a \mid \& S \mid S \! T \mid T \, b \\
T \rightarrow a \mid b \mid \varepsilon
$$

e. Indicate whether the following strings are in the language accepted by this grammar. If a string is in the grammar, show parse justifying it. For example, the string $\&\&ab$ is in the grammar, justified by the following parse:

$S \rightarrow \&\& \rightarrow \&\&S \rightarrow \&\&Tb \rightarrow \&\&ab$

i. (3 pts) $\&a\&b$

**Answer:** false (no way to get $\&a\&$ pattern)

ii. (3 pts) ab!

**Answer:** $S \rightarrow S!T \rightarrow Tb!T \rightarrow ab!T \rightarrow ab$

f. (3 pts) Is this grammar ambiguous? Why or why not?

**Answer:** Yes, because there are multiple ways to parse the string $\&a!b$:

$S \rightarrow S!T \rightarrow \&S!T \rightarrow \&a!T \rightarrow \&a!b$

$S \rightarrow \&S \rightarrow \&S!T \rightarrow \&a!T \rightarrow \&a!b$