CMSC330 Fall 2015 Midterm #1
12:30pm/2:00pm/5:00pm

Name: _____

Discussion Time:	**10am**	**11am**	**12pm**	**1pm**	**2pm**	**3pm**
TA Name (Circle):	**Adam**	**Maria**	**Chris**	**Chris**	**Michael**	**Candice**
	Amelia	**Amelia**	**Samuel**	**Josh**	**Max**	

Instructions

- 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 short-answer 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.

	Problem	Score
	PL Concepts	/13
	Ruby Regular Expressions	/10
	Ruby Execution	/18
	Ruby Programming	/14
	OCaml Typing	/15
	OCaml Execution	/15
	OCaml Programming	/15
	Total	/100

1. Programming Language Concepts (13 pts)

a) (3 pts) What is the difference between *dynamic typing* and *type inference*?

Dynamic typing means type checking that occurs at run-time. Type inference is done at compile time, to infer types.

b) (3 pts) What is the key difference between a *function pointer* and a *closure*?

Possible answers:

- ***A function pointer is just a pointer to code, whereas a closure also includes an environment.***
- ***A function pointer cannot be safely returned from another function if it refers to that function's variables; a closure can be.***

c) (2 pts) ***True*** or False: *Static typing* is compatible with *implicit declarations*.

d) (3 pts) Multiple choice: What do we mean when we refer to *first-class functions*?

a. These are functions that are implemented using objects; i.e., they have a class
b. First-class functions have the same status as other data, i.e., they can be created in and returned from (other) functions, and stored in data structures
c. These are functions that encode an object
d. These are things like Ruby code blocks

e) (2 pts) Suppose we have a variable `x` of type `int ref`. Circle those statements below that are true (you may circle either, neither, or both):

1) x is immutable
2) the storage that `x` points to is immutable

2. **Ruby Regular Expressions** (10 pts)

a. (3 pts) Give an English description of the strings matched by Ruby regexp `C?[msc]*330`

An optional C; followed by 0 or more m's, s's, and c's; followed by 330.

b. (4 pts) Circle which of the following strings matches the Ruby regexp `[aeiou][0-9]+$`

"who13" "o123o" ***"a2"*** "cmsc330"

c. (3 pts) Give a Ruby regexp that denotes the language of strings with any number of a's and an odd number of b's

a*b(a*ba*ba*)*

3. **Ruby Execution** (18 pts)

To the write of each Ruby code snippet below, indicate what happens when you run the code. If there is a run-time error write *FAIL*; otherwise, write what is printed out. Assume that the `nil` object prints as "`(nil)`" (and not the empty string).

a) (4 pts)

```
h = { 1 => "CS", 2 => "Math" }
h.keys.each{ |x| puts "#{h[x]}" }
```

Answer**: **CS\nMath\n

b) (3 pts)

```
a = [ ]
a[1] = 0
puts a["hello"]
```

Answer: FAIL (array index using string)

c) (3 pts)

```
a = [ ]
a[3] = 2
puts a
```

Answer: (nil)\n(nil)\n(nil)\n2\n

d) (4 pts)

```
ws = "1.23 4 hello fred"
x, y, z = ws.split(/\s/)
puts x+y
```

Answer: 1.234

e) (4 pts)

```
class Thing
  @@things = 0
  def initialize(name)
    @name = name
    @@things += 1
  end
  def self.get_things
    return @@things
  end
end
Thing.new("thing20");
Thing.new("thing6")
puts Thing.get_things
```

Answer: 2

4. **Ruby Programming** (14 pts)

On the next page is the initial Ruby implementation of a class `Set`, which implements a collection of distinct elements. This implementation contains a constructor (`initialize`) and method `add`. Extend the implementation with **any 2 of the following 3 functions** (7 pts each)

- **`member?(x)`** returns true if x is in the set, false otherwise
- **`elems`** returns a copy of the contents of the set as an array (order doesn't matter)
- **`union(x)`** returns a new set that contains all the elements in the receiving set and the elements in x

For full credit, do not change the existing methods or add methods beyond those required of you. If you can't figure it out without such changes, you can make them for partial credit.

Here is an IRB session with the class.

```
>> s1 = Set.new
=> #<Set:0x007fd0f38f5608 @s={}>
>> s1.add(1).add(1)
=> #<Set:0x007fd0f38f5608 @s={1=>true}>
>> s1.member?(1)
=> true
>> s2 = Set.new
=> #<Set:0x007fd0f38ffbf8 @s={}>
>> s2.add(1).add(2).add(2).elems
=> [1, 2]
>> s3 = Set.new
=> #<Set:0x007fd0f390f788 @s={}>
>> s3.add(3).add(4).add(2)
=> #<Set:0x007fd0f390f788 @s={3=>true, 4=>true, 2=>true}>
>> s4 = s1.union(s3)
=> #<Set:0x007fd0f503b100 @s={1=>true, 3=>true, 4=>true, 2=>true}>
>> s4.elems
=> [1, 3, 4, 2]
```

Answer to problem 4:

```
class Set

 def initialize
   @s = { }
 end

 def add(x)
   @s[x] = true
   return self
 end

 # put your member?, elems, and union methods below

def member?(x)
 return (@s[x] != nil)
end

def union(m)
 t = Set.new
 @s.keys.each { |element|
   t.add element
  }
 m.elems.each { |element|
  t.add element
 }
 t
end

def elems
 @s.keys
end




end
```

5. **OCaml Typing** (15 pts)

a) (3 pts) What is the type of the following OCaml expression?

```
(6, [1;2;3;4;5;6])
```

int * int list

b) (4 pts) What is the type of `foo` in the following OCaml definition?

```
let rec foo x y =
  match x with
    [] -> []
  | h::t -> if h = y then x else y::(foo t y)
;;
```

'a list -> 'a -> 'a list

c) (4 pts) Write an OCaml expression or definition of type `(int * string) list`

[(1,"hello")]

d) (4 pts) Write an OCaml expression or definition of type `(int -> int) -> int -> int`

Possible answers:

- ***fun f -> fun x -> let z = f 1 in x+z***
- ***let foo f x = (f 1)+x***

6. OCaml Execution (15 pts)

To the right of each code snippet, write what the variable `res` will contain after executing the code. Write *FAIL* if an exception is thrown.

a) (3 pts) ***Answer: 1***

```
let f (a,b) = a;;
let res = f (1,2);;
```

b) (4 pts) ***Answer: [12;10;17;20]***

```
let rec map f l = match l with [] -> [] | h::t -> (f h)::map f t
let clip x =
  if x < 10 then 10 else
  if x > 20 then 20 else x
;;
let cliplist l = map clip l;;
let res = cliplist [12;0;17;80];;
```

c) (4 pts) ***Answer: 102***

```
let g x y = fun b -> if x = b then b+x else b+y;;
let res = g 48 100 2;;
```

d) (4 pts) ***Answer: FAIL (match case missing)***

```
let rec trans f w =
  match w with
    ([x],[y],[z]) -> (f x, f y, f z);;
trans (fun x -> x+1) ([1],[2],[3;4]);;
```

7. **OCaml programming** (15 pts)

Below is some code provided in project 2b: the type `int_tree` of a binary search tree of integers, and the function `int_insert` that returns a new tree with an element added.

```
type int_tree =
    IntLeaf
  | IntNode of int * int_tree * int_tree

let rec int_insert x t =
  match t with
    IntLeaf -> IntNode(x,IntLeaf,IntLeaf)
  | IntNode (y,l,r) when x > y -> IntNode (y,l,int_insert x r)
  | IntNode (y,l,r) when x = y -> t
  | IntNode (y,l,r) -> IntNode(y,int_insert x l,r)
```

On the next page, **Implement any *three* of the following four functions**. (If you do all four, all will be graded, and the result scaled to be out of 15 points.)

1) `print_preorder t`, with type `int_tree -> unit`, prints out the contents of the tree `t` in preorder (i.e., the node, then the left tree's contents, then the right tree's contents). You can use the `print_int` function for printing integers.

2) `min_node t`, with type `int_tree -> int`, returns the minimum element of the tree, or throws exception `Invalid_argument "min_node"` if the tree is empty. Should run in time *O(height of the tree)*.

3) `is_emptytree t`, with type `int_tree -> bool`, returns `true` of the tree is empty, and `false` otherwise.

4) `sum t`, with type `int_tree -> int`, returns the sum of all the elements that appear in `t`.

Here are some example uses:

```
let tr = int_insert 7 (int_insert 2 (int_insert 5 IntLeaf));;
print_preorder tr;;           (* prints 527 *)
let x = min_node tr;;         (* x contains 2 *)
let y = is_emptytree tr;;     (* y contains false *)
let w = is_emptytree IntLeaf; (* w contains true *)
let z = sum tr;;              (* z contains 14 *)
```

Answer to problem 7 here

```
let rec print_preorder t =
  match t with
      IntLeaf -> ()
    | IntNode (y,l,r) -> print_int y; print_preorder l; print_preorder r
;;

let rec min_node t =
  match t with
      IntLeaf -> invalid_arg "min_node"
    | IntNode (y,IntLeaf,_) -> y
    | IntNode (y,l,_) -> min_node l;;

let rec is_emptytree tr =
  match tr with IntLeaf -> true | _ -> false;;

let rec size t =
  match t with
      IntLeaf -> 0
    | IntNode (y,l,r) -> 1 + size l + size r
;;
```