CMSC330 Spring 2017 Midterm #1
Solution

Name (PRINT YOUR NAME as it appears on gradescope):

_____

Discussion Time (circle one) **10am 11am 12pm 1pm 2pm 3pm**

Discussion TA (circle one) **Aaron Alex Austin Ayman Daniel Eric**
Greg Jake JT Sam Tal Tim Vitung

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 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.

	Problem	Score
1	Programming Language Concepts	/12
2	Regular Expressions	/12
3	Ruby execution	/10
4	Ruby Programming	/18
5	OCaml Typing	/12
6	OCaml Execution	/18
7	OCaml Programming	/18
	Total	/100

1. Programming Language Concepts (12 pts)

a) (3 pts) Which of the following statements are true of higher-order functions? **Circle all that apply.**

i) They are functions that run in higher than O(n) time
ii) They are functions that can return other functions
iii) They are necessary for object-oriented programming
iv) They are functions that receive other functions as arguments
v) map and fold are examples of higher order functions

b) (3 pts) Show the contents of the closure for g after executing the following code

```
let f = (fun x-> (fun y-> (fun z -> x * z)));;
let g = f 10 20;;
```

Code	Environment
fun z -> x*z	**x=10** **y=20**

c) (6 pts) Circle T (true) or F (false) for the following statements (1pt each)

i) T / **F** OCaml tuples are homogeneous.
ii) **T** / F With immutable state, aliasing is irrelevant.
iii) T / **F** Methods can be overloaded in Ruby.
iv) T / **F** Ruby is a compiled language.
v) **T** / F For OCaml, type inference happens at compile-time.
vi) T / **F** Ruby has static typing.

2. Regular Expressions (12 pts)

a) (3 pts) Circle **all** of the strings that match the Ruby regular expression `/^\d{3}.\d{4}$/`

i) 732.7444
ii) 908-5490
iii) 201-901
iv) ddd.dddd

b) (3 pts) What is the output of the following Ruby code?

```
"Why was 6 afraid of 7?" =~ /\d\s(\w+).*(\d)/
puts $1
puts $2
```

afraid
7

c) (3 pts) Write the output of the following code. (Recall that `a.inspect` gives the representation of a as it would appear in source code, e.g. `[1,2,3].inspect` is `"[1,2,3]"`.)

```
s = "To be, or not to be!"
a = s.scan(/(\S+) (\S+)/)
puts a.inspect
```

[["To","be,"],["or","not"],["to","be!"]]

d) (3 pts) A phone company needs to match serial numbers. The format is either 2 or 4 alphabetic characters followed by 1 to 3 numeric characters. Write the a Ruby regular expression which matches these serial numbers.

Example valid Inputs	Example invalid Inputs
aT7	76n
op115	k997
uaqt345	abcde123

/^([A-Za-z]{2}|[A-Za-z]{4})\d{1,3}$/

3. Ruby Execution (10 pts)

Write the **output** of the following Ruby code. If there is an error, then write `FAIL`.

a) (3 pts)

```
arr = [true,nil,false,0]
arr.each do |x|
      if x then
            puts “Ruby”
      else
            puts “Crystal”
      end
end
```

output:

Ruby
Crystal
Crystal
Ruby

b) (4 pts)

```
class JellyBean
  def initialize
    @@flavors = ["blue", "red", "purple", "yellow"]
  end
  def foo(f)
    @@flavors.push f
  end
  def flavors
    @@flavors
  end
end
j = JellyBean.new()
k = JellyBean.new()
j.foo("orange")
k.foo("rainbow")
puts j.flavors.inspect
```

[“blue”,”red”,”purple”,”yellow”,”orange”,”rainbow”]

c) (3 pts)

```
h = { "a"=>4, "b"=>3, "c"=>2, "d"=>1 }
r = h.values.select { |x|
    x % 2 == 0
```

```
}
r.sort!
puts r.inspect
```

```
[2,4]
```

4. Ruby Programming (18 pts)

For this question, you implement a "**memory game**" in class `MemoryGame`. In the game, a player picks two cards (each an object of class `Card`) from a deck (an object of class `Deck`). If the two cards have the same face value (regardless of suit), they are marked as matched. Players may not match cards that have been matched previously. Players can play until every card is matched. We provide starter code for the `Card` and `Deck` classes below. Notice that:

- Suits are strings:`"Spades", "Clubs", "Hearts", "Diamonds"`
- Values are strings or integers: `"Ace", "King", "Queen", "Jack", 2, 3, 4, 5, 6, 7, 8, 9, 10`
- the deck is shuffled when it is created.

```
class Card
  attr_accessor :suit, :value
  def initialize(s, v)
      @suit = s
      @value = v
  end
  ... # you will add a method here
end
```

```
class Deck
  def initialize
      @cards = []
      suits = ["Spades", "Clubs", "Hearts", "Diamonds"]
      values = ["Ace", "King", "Queen", "Jack", 2, 3, 4, 5, 6, 7, 8, 9, 10]
      values.each {|v|
            suits.each {|s|
                  c = Card.new(s,v)
                  @cards.push(c)
            }
      }
      @cards.shuffle!         # randomly permutes the cards
  end
... # you will add two methods here
end
```

a) (6 pts) Write the `self.from_string(s)` static method for the `Card` class. This method receives a string in "value suit" format (for example: "10 Clubs", "Queen Spades"), and creates a card with that value and suit. It returns `nil` if the given string is not a valid value and suit.

```
  def self.from_string(s)

if s =~ /^([2-9]|10|Ace|King|Queen|Jack) (Spades|Clubs|Hearts|Diamonds)$/
      return Card.new($2,$1)
 end

  end       # end of self.from_string method
end         # end of Card class

  ...
```

b) (2 pts) Write the method `cardAt(n)` for the `Deck` class. It returns the card at position `n` (where indexing starts at 0). Returns `nil` if there is no card at that position.

```
  def cardAt(n)

      return @cards[n]


  end       # end of Deck's cardAt method
```

c) (2 pts) Write the method `numCards` for the `Deck` class, which returns the number of cards in the deck

```
  def numCards


      return @cards.count


  end             # end of Deck's numCards method
end               # end of Deck class
```

d) (8 pts) Write the `MemoryGame` class, which implements the “memory game” described above. The `MemoryGame` constructor creates a fresh `Deck` and stores it in field `@d`. You must implement method `match(n1,n2)`, which checks if two cards at positions `n1` and `n2` in `Deck @d` have the same values (irrespective of the suit). You may need to add code to the constructor.

`match(n1,n2)` returns `true` if

- the cards at positions n1 and n2 in Deck `@d` have the same value and
- have not been matched before (i.e., `match(n1,n2)` has not previously returned `true`)

`match(n1,n2)` returns `false` if

- `n1 == n2`, or
- `n1` or `n2` are not valid positions in the `Deck @d`, or
- the cards at `n1` and `n2` do not have the same value, or
- the cards at `n1` and `n2` were part of a previous match

```
class MemoryGame
  def initialize  # hint: add a data structure to keep track of matched cards
      @d = Deck.new
      @matched = Hash.new(false)

  end       # end of initialize

  def match(n1,n2)
      if (n1 == n2 || n1 < 0 || n2 >= @d.numCards || @matched[n1] ||
      @matched[n2] || @d.cardAt(n1).value != @d.cardAt(n2).value)
            return false
      end

      @matched[n1] = true
      @matched[n2] = true

      return true




  end       # end of match method
end         # end of MatchGame class
```

5. OCaml Typing (12 pts)

a) Determine the types of the following OCaml expressions

1. (2 pts) `[(1, 3.14, "hello")]`

   Solution: (int * float * string) list

2. (2 pts) `fun a b c -> if a then (b a) else (c a)`

   Solution: bool -> (bool -> 'a) -> (bool -> 'a) -> 'a

3. (2 pts)
```
let f x m =
  match m with
| [] -> x
| h::_ -> x + h
```

   Solution: int -> int list -> int

b) Provide an OCaml expression that has the given type. (You may not use type annotations.)

1. (3 pts) `int -> int list -> int list`

   Possible Solution: `fun x y -> (x+1)::y`

2. (3 pts) `` `a * `b -> (`b -> `a) -> `a list ``

   Possible Solution: `fun (x,y) z -> (z y)::[x]`

6. OCaml Execution (18 pts)

Write **what the following expressions evaluate to**. If there is an error, write ERROR and circle the error in the code. Below are implementations of `map` and `fold` below for reference (they are equivalent to the standard `List.map` and `List.fold_left` functions).

```
let rec fold f a l =
  match l with
  | [] -> a
  | h::t -> fold f (f a h) t
;;

let rec map f l =
  match l with
  | [] -> []
  | h::t -> let r = f h in r::(map f t)
;;
```

a) (3 pts.)

```
let cmsc f x y = if (f x y = 0) then 1 else 0 in
cmsc (fun a b -> a - b) 4 4
```

Solution: 1

b) (3 pts.)

```
map (fun x -> x + 2) (6::[1;4;5])
```

Solution: [8;3;6;7]

c) (4 pts.)

```
let f l = fold (fun a x -> a || x) false l in
if f [false; true; false] then 1 else 0
```

Solution: 1

d) (4 pts.)

```
let x = 7 in
let add a b = a + b in
let addto = add x in
let x = 40 in
(addto 10, add x 10, addto x)
```

Solution: (17,50,47)

e) (4 pts.)

```
let f =
  let c = ref 0 in
  fun x -> c := !c + x; !c in
(f 2) + (f 4)
```

Solution: 10

7. OCaml Programming (18 pts)

Implement the following functions. You are allowed to add helper functions for any of these problems if you like.

a) (6 points) Define a function `repeat : int * 'a -> 'a list` that takes a non-negative integer `n` and a value `x` and returns a list containing `n` copies of `x`. (You may implement this with **fold** or **map**, if you wish.) Examples:

```
repeat (0,1) = [];;
repeat (2,"hello") = ["hello"; "hello"];;
```

```
let rec repeat a = match a with
      (0,v) -> []
      | (n,v) -> v::(repeat ((n-1),v))
;;
```

b) (6 points) Define a function `unpack l : (int * 'a) list -> 'a list` that takes a list of of pairs and returns a list of the second elements of those pairs, where those elements appear as many times as indicated by the first elements. This is called “run length” format.

(cont’d next page)

You may *not* write a recursive function; instead you *must* use `fold` and/or `map` (whose definitions are given in problem 6, above). If you wish, you may use any function in the `List` or `Pervasives` library, or anywhere else in this exam, to help.

```
unpack [(1,2);(3,1);(2,5)] = [2;1;1;1;5;5];;
unpack [(2,"hello");(0,"fourteen")] = ["hello";"hello"];;
```

```
let unpack lst = fold (fun a h -> a@(repeat h)) [] lst;;
```

c) (6 points) Consider the following variant type definition `exp`, which represents arithmetic expressions that involve integer constants, multiplication, and addition:

```
type exp =
  | Int of int
  | Mult of exp * exp
  | Plus of exp * exp
```

Define a function `eval : exp -> int` that takes an expression and returns the `int` it evaluates to. Examples:

```
eval (Int 3) = 3;;
eval (Plus(Int 1,Int 2)) = 3;;
eval (Mult(Plus(Int 1,Int 2),Int 3)) = 9;;
eval (Mult(Int 2,Plus(Int 1,Int 2))) = 6;;
```

```
let rec eval ex = match ex with
      Int(i) -> i
      | Plus (a,b) -> (eval a) + (eval b)
      | Mult (a,b) -> (eval a) * (eval b)
;;
```

THIS PAGE INTENTIONALLY LEFT BLANK