

# CMSC330 Spring 2016 Midterm #1

## 9:30am/12:30pm/3:30pm

Name: \_\_\_\_\_

Discussion Time: 10am 11am 12pm 1pm 2pm 3pm  
TA Name (Circle): Adam Anshul Austin Ayman Damien  
Daniel Jason Michael Patrick William

### 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 | <u>15</u> |
| 2 | Regular Expressions           | <u>10</u> |
| 3 | Ruby execution                | <u>10</u> |
| 4 | Ruby Programming              | <u>15</u> |
| 5 | OCaml Typing                  | <u>16</u> |
| 6 | OCaml Execution               | <u>18</u> |
| 7 | OCaml Programming             | <u>16</u> |
|   | Total                         | /100      |



## 1. Programming Language Concepts (15 pts)

- a) (2 pts) (True / False) The = operator in OCaml performs structural comparison
- b) (3 pts) In each part, circle the validity of the statement given the expression:  
**let comb x lst = (x + 5)::lst ;;**
- (True / False) OCaml uses type inference to recognize that x must be an int.  
(True / False) comb 2 [8.0;9.0] is valid. Implicit type conversion will cast x + 5 to a float  
(True / False) comb 2 will produce a closure that contains x -> 2.
- c) (2 pts) Which of the following is NOT true about OCaml (circle all that apply)?
- i) It is statically typed
  - ii) types are inferred
  - iii) it doesn't have garbage collection
  - iv) it evaluates x before it evaluates y in let x = <code> in y
- d) (2 pts) Which of these is true about Ruby (circle all that apply)?
- i) Ruby is a compiled scripting language.
  - ii) Ruby uses dynamic typing.
  - iii) Ruby is a functional programming language.
  - iv) Ruby uses explicit declarations.
- e) (2 pts) Which of these is a difference between Ruby and Java (circle all that apply)??
- i) Ruby is statically typed while Java is not.
  - ii) Java is object-oriented while Ruby is not.
  - iii) Java is a scripting language while Ruby is not.
  - iv) Ruby supports implicit declarations while Java does not.
- f) (2 pts) Which of the following features help OCaml reduce the probability of errors occurring at runtime (circle all that apply)?
- i) Static Typing
  - ii) Currying
  - iii) Modules
  - iv) Type Inference
- g) (2 pts) What is a typical advantage of a compiled language over an interpreted language (circle all that apply)?
- i) Improved performance
  - ii) Shorter development time
  - iii) Manual Memory Management
  - iv) Write Once, Run Anywhere (ie. The same code will "just work" on most platforms)
  - v) Improved type safety

## 2. Regular Expressions (10 pts)

1. (3 pts) Describe the strings matched by the following regex: `/^(0(0|1)*0)|(1(0|1)*1)$/`

2. (3 pts) What is the output of the following Ruby code:

```
if "Email: eggplant_Lover512@veggies.com" =~ /\w+(@\w+)\.com/ then
  puts $1
else
  puts "No match"
end
```

3. (4 pts) Write a regular expression that matches with **types** of OCaml tuples that only consist of ints and floats. Note that the `*` symbol needs to be escaped in regexes. You may assume that the input strings will contain no whitespace, and have at least two types.

Example Matches: `"int*float"` `"int*int*int"` `"float*int*float*int"`

## 3. Ruby Execution (10 pts)

1) (4 pts) What is the output (if any) of the following Ruby programs? If the code does not execute, write FAIL.

```
h = {"hello" => "salut", "bye" => "au revoir"}
y = h.keys.collect {|k| k + "->" + h[k]}
y.sort!
puts y[0]
```

### Answer

2) (3 pts)

```
class Animal
  def initialize(height, weight)
    @@height = height
    @weight = weight
  end
  def measure()
    return @@height + @weight
  end
end

giraffe = Animal.new(1,2)
elephant = Animal.new(3,4)

puts giraffe.measure()
puts elephant.measure()
```

**Answer**

3) (3 pts)

```
a = []
a[1] = "Strawberry"
a[3] = "Tomato"
a.each { |x|
  if x
    puts "Fruit"
  else
    puts "Veggie"
  end
}
```

**Answer**

#### 4. Ruby Programming (15 pts)

When applications or websites require password authentication, they must store the passwords associated with each user. However, storing the actual passwords without any encryption is a critical security hazard. Instead, the passwords are securely *hashed (encrypted)*, so that even if the database is compromised, attackers cannot read the passwords.

A password hash (encryption) is valid according to our (bad) algorithm if:

- It is 10-character long
- It includes no whitespace
- The first character is 'h'
- The last two characters are '=='
- It contains at least one uppercase letter
- It contains at least one number

You are writing functions for a class named HashReader. It will read from text file, consisting of username and password\_hash pairs in the form of <username>;<password\_hash> . You can assume

- all the username/password\_hash pairs are in valid form
- each line contains exactly one pair
- all the usernames are distinct and can not contain semicolons.

Implement the following functions (5 pts each):

**initialize(filename):** read all the user/password\_hash data from text file, and store it in an appropriate data structure. In Ruby `IO.foreach` opens a file calls the given block for each line it reads and closes the file afterwards. You don't have to worry about closing the file. For example:

```
IO.foreach("file") { |line| puts line }
```

**is\_valid(password\_hash):** Given a password\_hash, return true if the hash\_value meets the requirements for a valid hash given above. Otherwise, return false. **You have to use regex for validation.**

**iter():** Takes in a code block and passes each username / password\_hash pair to the block. Iterate over each username / password. **You have to use yield to implement iter.**

**users.txt:**

```
alice;h9INffmk==
bob;h9INffmk==
cathy;h12345fH==
dinosaurav;h867F309==
```

**Example usage:**

```
a = HashReader.new("users.txt")
puts a.is_valid("This is a password==")
a.iter { |username, hash|
  puts "#{username};#{hash}"
}
```

**Output:**

```
false
alice;h9INffmk==
bob;h9INffmk==
cathy;h12345fH==
dinosaurav;h867F309==
```

```
class HashReader
```

End

## 5. OCaml Typing (16 pts)

1) (6 pts) Without using type annotations, write an expression for each of the following types:

i) `int list -> int`

ii) `'a -> 'a -> bool`

iii) `(int -> int) -> (int -> int) -> int`

2) What is the type of the following function given the following data type?

```
a) (4 pts)
type 'a atree =
  Leaf
  | Node of 'a * 'a atree * 'a atree
;;
let rec apply t f=
  match t with
  | Leaf -> t
  | Node(x, l, r) -> Node(f x, apply l f, apply r f)
;;
```

b) (3 pts) `let mult_and_div x y = (x *. y, x /. y)`

c) (3 pts) `[(4, "Hello");(5, "World")]`

## 6. OCaml Execution (18 pts)

- 1) (5 pts) What is `r` bound to at the end of the following code? If there is an error, mark the line and write ERROR with a brief explanation.

```
let v = 2;;
let f a b = a * b;;
let f' = f v;;
let v = 5;;
let r = (f' 10, f v 10);;
```

- 2) (5 pts) What does the following statement evaluate to? Refer to the definition of `fold` provided in question 7, next page.

```
fold (fun (a1,a2) v -> (a1 + v*2, a2 + v/2)) (0,0) [2;4;7];;
```

- 3) (4 pts) What is `r` bound to at the end of the following expression? Use `fold` provided in question 7, next page.

```
let rec apply_two d f1 f2 = f2 (f1 d);;
let data = [1;2;3];;
let r = apply_two data (map (fun x->10*x)) (fold (fun a x->a+x) 100);;
```

- 4) (4 pts) We define a function `inc` as follows:

```
let inc =
  let x = ref 0 in
  fun y -> x := !x + y; !x
```

What is the value of the following expression? If there is an error, write ERROR and give a brief description of the problem.(Use the `map` given in Question 7, next page)

```
map inc [1; 2; 3; 4]
```

## 7. OCaml programming (16 pts)

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

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

Implement any two of the following three problems. (If you do all three, all will be graded, and the result scaled to be out of 16 points)

- 1) Use **fold and map** to write a function, `f`, that takes one argument, a list of list of ints, and outputs a list of ints, replacing each of the list of ints inside the original list with their respective sums. Eg: `f [[1;2];[];[4,5,3]]` evaluates to `[3;0;12]`. Ensure that order of sums conforms to the order of int list in the original list of list of ints.

- 2) Partition: Write a recursive OCaml function which takes a list and a predicate function and returns a tuple of 2 lists. The first list is all of the elements of the argument list which satisfy the predicate, while the second list is all of the elements which do not satisfy the predicate. The type of partition is the following:

`'a list -> ('a -> bool) -> 'a list * 'a list`

e.x. `partition [-1;-5;2;-6;5] (fun x -> if x > 0 then true else false)` returns  
`([2;5], [-1;-5;-6])` or `([-6;-5;-1], [2;5])`

- 3) Define a recursive function `merge` that takes two sorted lists as arguments and returns a sorted list that is the result of merging the two lists.