

CMSC330 Spring 2018 Midterm 1

9:30am/ 11:00am/ 3:30pm

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

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	/10
2	Ruby Regular Expressions	/10
3	Ruby Execution	/17
4	Ruby Programming	/18
5	OCaml Typing	/14
6	OCaml Execution	/13
7	OCaml Programming	/18
	Total	/100

1.[10 pts] Programming Language Concepts

Circle your answer

- A. Tuples in OCaml are similar to structs in C in that they are both fixed-sized collections of heterogeneous data. (T / F)
- B. Ruby has type inference for its variables. (T / F)
- C. In dynamically typed languages, type errors may go unnoticed if they are inside rarely used conditional branches. (T / F)
- D. A let...in expression in Ocaml is used to define a named local expression. (T / F)
- E. Because of dynamic type checking, Ruby allows programs with type errors to run. (T / F)
- F. Ruby arrays can hold different objects and dynamically resizable. (T / F)
- G. Both Procs in Ruby and functions in OCaml have “first class” status; e.g., they can be passed to and returned from methods/functions. (T / F)
- H. If two objects are structurally equal, they must be physically equal too. (T / F)
- I. A closure consists of function code and bindings for its free variables. (T / F)
- J. Compiled languages typically run slower than interpreted languages because of the extra overhead of converting source code to machine code at runtime. (T / F)

2. [10 pts] Ruby Regular Expressions

A. (2 pts) What is the output of the following?

```
"I am Groot!" =~ /\w+ \w+ (\w+).$/
```

```
puts $1
```

Answer:

B. (4 pts) Write a Ruby regular expression that matches dates of the form MM/DD/YYYY. MM and DD can be ONE or TWO digits (they do not need to be valid months or days respectively, see examples below). YYYY must be exactly FOUR digits. The regex must match the string exactly.

Examples:

7/4/1776

6/23/1998

12/25/0000

5/16/2019 (This is the date of your final!)

99/99/9999 (This is valid format, although not a valid month or day)

Answer:

C. (4 pts) Circle all those strings that *match* the regular expression

`/[A-Z]+[a-z]*:\s?[0-5]+$/`. Put another way, circle each string `s` for which

`s =~ /[A-Z]+[a-z]*:\s?[0-5]+$/` does not return `nil`.

123Anwar: 12

eastman: 34

CMSC:330

Mike: 56

3. [17 pts] Ruby Execution

Write the printed output of the following code snippets

1. (3 pts)

```
x = [1, 1, 2, 3, 5]
puts x[0]
puts x[5]
y = [1, 1, 2, 3, 5]
puts x == y
```

Answer:

2. (3 pts)

```
grades = {"Alice" => 0, "Bob" => 4, "Chris" => 3 }
if grades["Alice"] then
  grades["Alice"] = 2
end
puts grades["Alice"]
sum = 0
grades.keys.each {|k| sum = sum + k.length }
puts sum
```

Answer:

3. (3 pts)

```
def math(x)
  if x % 2 == 0
    puts yield(x)
  else
    puts yield(x+1)
  end
end
math(10) {|z| z+10}
math(3) {|z| z*3}
math(0) {|z| z-4}
```

Answer:

4. (4 pts)

```
h = { 1 => "cat", 2 => "squirrel", 3=>"chicken" }
x = h.keys.collect{|k| h[k] }
puts x[1]
```

Answer:

5. (4 pts)

```
class ToolchainManager
  @@x = []
  def initialize(version)
    @@x.push(version)
    @count = 1
  end
  def update()
    @@x.push(@count)
    @count += 1
  end
  def to_s
    @@x.length.to_s + "," + @count.to_s
  end
end
cargo = ToolchainManager.new("1.33.0")
puts cargo
cargo.update()
puts cargo
cargo.update()
puts cargo
cult = ToolchainManager.new("1.33.5")
puts cult
```

Answer:

4. [18 pts] Ruby Programming

Implement an `HashStack` class. `HashStack` is like a hash, but if you add a mapping for a key that's already in the `HashStack`, it remembers the old mapping and pushes the new one, like a stack. When you remove an entry, the old mapping is restored.

(7pts) `insert(k, v)` adds a mapping from `k` to `v` in your `HashStack` instance. If a mapping for `k` already exists, the new mapping overrides it, but the old mapping is remembered. Return `nil` for a fresh mapping; if overriding an existing mapping, return the *old* value.

(7pts) `remove(k)` removes the most recent mapping for `k`, returning the value component of it. If a mapping for `k` doesn't exist, return `nil`.

(4pts) `find(k)` returns the value most recently mapped to by `k`. If a mapping for `k` doesn't exist, return `nil`. Leaves the existing mapping(s) in place.

Here is an example session with a `HashStack`.

```
irb(main):003:0> m = HashStack.new
=> #<HashStack:0x00007ff518868f70 @h={}>
irb(main):004:0> m.insert("a",2)
=> nil
irb(main):005:0> m.insert("b",3)
=> nil
irb(main):006:0> m.find("b")
=> 3
irb(main):008:0> m.insert("a",3) # overrides existing mapping
=> 2
irb(main):009:0> m.find("a")
=> 3
irb(main):010:0> m.remove("a")
=> 3
irb(main):011:0> m.find("a")
=> 2
irb(main):012:0> m.remove("a")
=> 2
irb(main):013:0> m.find("a")
=> nil
irb(main):015:0> m.remove("b")
=> 3
```

```
class HashStack
  def initialize
    @h = {}
  end
  def insert(k, v)
```

```
end
  def remove(k)
```

```
end
```

```
  def find(k)
```

```
end
```

5. [14 pts] OCaml typing

A. (6 pts) Write an expression of the following type **without using type annotations**

a. `float * (float list) * string`

b. `float -> float list -> float list`

c. `int -> 'a -> 'a`

B. (8 pts) Give the type that OCaml will infer for `f` in each of the following. If there is a type error, circle where the issue is and explain

a. `let f x = x * 4`

b. `let f a b = (a::b)::[b]`

c. `type vector = { x : int; y : int }
let f v a = v.x > a`

d. `type int_option = Nothing | Something of int`

```
let f = fun a -> match a with  
  Nothing -> 0  
  | Something i -> []
```


6. [13 pts] OCaml Execution

```
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 -> (f h)::(map f t)
```

Give the value of the final expression in each of the following. If there is a type error, show where. If an exception is raised, say what it is.

A. (2 pts)

```
let rec f l =
  match l with
  | [] -> []
  | h1::h2::t -> (h1*h2)::(f t);;
```

f [1;2;3;4;5;6]

Answer:

B. (2 points)

```
let f2 f x y =
  if (f x y) = 0 then 1
  else 0;;
```

f2 (fun a b -> a*b) 10 0

Answer:

C. (3 points)

```
let f (m, s) x =
  if (x > m) then (x, s+x)
  else (m, s+x);;
```

fold f (0,0) [10;3;8;0]

Answer:

D. (2 points)

```
let f a = a * 2;;  
map f [1; 2; 3; 4; 5]
```

Answer:

E. (4 points)

```
type float_tree =  
  Leaf  
  | Node of float_tree * float_tree * float;;  
  
let t1 = Leaf ;;  
let t2 = Node(Node(Leaf, Leaf, 5.0), Leaf, 4.0) ;;  
let t3 = Node(Leaf, Leaf, 3.0) ;;  
let tree_func t =  
  match t with  
  | Leaf -> false  
  | Node(l,r,f) -> l = Leaf && r = Leaf;;  
  
map tree_func [t1;t2;t3]
```

Answer:

7. [18 pts] OCaml Programming

```
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 -> (f h)::(map f t)
```

1. (5 pts) Write a function `partial_sum` with type `float -> float list -> float`. The `partial_sum` function should take a minimum value and a list and then return the sum of all of the values in the list that are greater than or equal to the provided minimum value. For full credit, you **must use map and/or fold** (in a non-superfluous way) to implement `partial_sum`.

Examples:

```
partial_sum 3.1 [] = 0.0
```

```
partial_sum 2.4 [5.3; 2.4; 1.0] = 7.7
```

Answer:

2. (6 pts) At your favorite Mexican Grill, burrito bowls can have three types - Veggie, Chicken or Steak. An order can either be some kind of bowl or a bag with a pair of orders in it, expressed as the order type as follows:

```
type order =
  Veggie_bowl
  | Chicken_bowl
  | Steak_bowl
  | Bag of order * order
```

Write a function `is_veggie` of type `order -> bool` that computes whether an order consists entirely of vegetarian items.

Examples:

```
is_veggie Veggie_bowl = true
```

```
is_veggie (Bag(Veggie_bowl,Veggie_bowl)) = true
```

```
is_veggie (Bag(Veggie_bowl,Bag(Veggie_bowl,Steak_bowl))) = false
```

```
is_veggie (Bag(Bag(Veggie_bowl,Veggie_bowl),Bag(Veggie_bowl,Veggie_bowl)))
```

Answer:

3. (7 pts) Write a function `bag_order` that takes an order `list` and produces a single order, containing all of the orders in the list. If given an empty list, throws exception `Invalid_argument "empty"`

Examples:

```
bag_order [Veggie_bowl] = Veggie_bowl
```

```
bag_order [Veggie_bowl; Chicken_bowl] = Bag(Veggie_bowl, Chicken_bowl)
```

```
bag_order [Veggie_bowl; Chicken_bowl; Steak_bowl] =  
    Bag(Veggie_bowl, Bag(Chicken_bowl, Steak_bowl))
```

Answer :

Next question is optional and worth zero point.

4. (0 pts) Write a function `flat_bag` that takes an order and “flattens” it, so that for any Bags in the order, the left component of the Bag is never itself a Bag. The order of the non-bag elements should be the same. *Hint:* You will want to use the `bag_order` function to help.

Examples:

```
let b = (Bag (Bag(Veggie_bowl, Veggie_bowl),Steak_bowl));;
flat_bag b = Bag (Veggie_bowl, Bag (Veggie_bowl, Steak_bowl));;
flat_bag (Bag(b,b)) =
  Bag (Veggie_bowl, Bag (Veggie_bowl, Bag (Steak_bowl,
    Bag (Veggie_bowl, Bag (Veggie_bowl, Steak_bowl)))));;
```

Answer: