CMSC330 Fall 2019 - Midterm 1

First and Last Name (PRINT): _____

9-Digit University ID: _____

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.

1. Programming Language Concepts	/ 10
2. Ruby Regular Expressions	/ 10
3. Ruby Execution	/ 17
4. Ruby Programming	/ 18
5. OCaml Typing	/ 15
6. OCaml Execution	/ 14
7. OCaml Programming	/ 16
8. Total	/ 100

Please write and sign the University Honor Code below: **I pledge on my honor that I have not given or received any unauthorized assistance on this examination.**

_____

_____

_____

Signature: _____

[10pts] Programming Language Concepts

Circle your answer. Each question is 1 point.

T   F   qsort in C is a higher order function because it takes a function pointer as an argument

T   F   In OCaml, `[1]::[2]` is equivalent to `[1;2]`.

T   F   OCaml type inference occurs at runtime.

T   F   In Ruby, `x = "apple"; y = x;` is an example of a reference copy.

T   F   OCaml tuples are homogeneous.

T   F   Structural equality implies physical equality.

T   F   For a statically-typed language, you have to specify the type of variables when declaring them.

T   F   Functions in OCaml are first class.

T   F   Ruby supports implicit variable declarations.

T   F   Ruby code blocks are first class.

[10pts] Ruby Regular Expressions

1   (2pts)   Circle the string(s) that match the following Ruby regular expression:

```
/[a-zA-Z]*\d?@cs.umd.edu$/
```

jane@cs.umd.edu   Rogereastman330@umd.edu

FirstnameLastname@cs.umd.edu   TA3@cs umd edu

2   (4pts)   What is the output of the following Ruby program?

```
"Final exam is 12-11-2019!" =~
    /^Final exam is (\d+)-([0-9]{1,2})-(\d*.)$/
puts "#{$2}th of December, #{$3}!!"
```

3   (4pts)   A movie theater is receiving reservations from a file, and you need to help them write a Ruby regular expression to make sure the lines are well formed. Write a Ruby regular expression of the form "id: R###, guests: N". The reservation id consists of the uppercase letter 'R' followed by three consecutive digits (here # is a single digit). N is any positive number of guests (0 is not a valid number of guests). The line must match exactly.

Examples of valid lines:
"id: R001, guests: 2"
"id: R002, guests: 13"
"id: R999, guests: 999"

Examples of lines that should NOT match:
"id: R001,   guests: 2"
"id: R002, guests: one"
"id: R9999, guests: 999"
"hello id: R999, guests: 999"

[17pts] Ruby Execution

Write the output of the following Ruby code. If there is an error, then write **ERROR**. If nil is printed, write **NIL**, not the empty string. **Hint: select** invokes the block by passing in the elements of the list (in order), and then returns an array containing those elements for which the block returned a true value.

1   (3pts)

```
x = []
x[4] = 5
puts x[6]
x.unshift(x.pop())
x.push("a")
x.each { |a|
    puts a
}
```

OUTPUT:

2   (3pts)

```
x = [1, 2, 3, 4]
x.collect! { |a|
    a = 2*a + 1
}
puts x
```

OUTPUT:

3   (3pts)

```
x = { 1 => "one", 2 => "two",
    3 => "three", 4 => "four"}
y = x.values.select { |a|
    a.length > 3
}
y.sort!
puts y
```

OUTPUT:

4   (4pts)

```
def newFunc(x)
    if x > 10
        puts yield x
    else
        puts yield (x / 2)
    end
end


newFunc(10) { |a| a * a }
newFunc(13) { |b| 2 * (b - 1) }
```

OUTPUT:

5   (4pts)

```
def apply(acc, elem)
    elem.size().times do |i|
        acc = yield(acc, elem[i])
    end
    acc
end

puts apply([], [3,5,7,9]) {|a,e| a.prepend e}.to_s
puts apply(0, [1,2,3,4,5,6]) {|a,e| if e % 2 then e + a else a end}
```

OUTPUT:

[18pts] Ruby Programming

Implement a shopping list class. As defined by the initialize method, a shopping list should be represented as a hash. DO NOT modify the initialize method in any way. The three methods you need to provide an implementation for are described below.

1   (4pts)   `add_item(name, quantity)`: This method should add an item with the given name and quantity into the shopping list. You can assume the name will always be a String and the quantity will always be an Integer. If the name of the item already exists in the shopping list, add the specified quantity to its current quantity. This method should RETURN nil.

2   (7pts)   `remove_item() {|name, quantity| block}`: This method should remove all items from the shopping list that make the code block evaluate to a truthy value. The passed in code block accepts as an argument the name and quantity of an item in the shopping_list. The state of the @shopping_list should be changed by the removal of the affected items. This method should RETURN the number of items that were removed. **Hint: use delete(key) to delete a key value pair from the shopping list. Also, a code block returns the value of its last expression as its result.**

3   (7pts)   `prune_shopping_list(item_prices, item_budget)`: Given a hash of the item_prices that maps an item's name to its price, and a hash item_budget that maps an item's name to its Integer budget value (every item name in the shopping list is guaranteed to be in both hashes and mapped to a non nil value), generate a new shopping list that contains only the items whose cost (i.e. quantity * price) is less than or equal to the amount budgeted for the item. RETURN the new shopping list. DO NOT change the state of @shopping_list in the process of writing this method.

Example Usage:

```
list = ShoppingList.new()
list.add_item("orange", 2)
list.add_item("orange", 5)      (* The quantity of orange is now 7 *)
list.add_item("banana", 10)
list.add_item("guava", 13)
list.add_item("plantains", 5)
```

The below line removes orange and plantains from the hash and returns 2
```
list.remove_items() { |name, quantity| quantity == 7 }
```

The below line returns a new shopping list with banana as the only item
```
list.prune_shopping_list({"banana" => 2, "guava" => 3},
    {"banana" => 20, "guava" => 25})
```

```ruby
class ShoppingList

    def initialize()
        @shopping_list = {}
    end

    def add_item(name, quantity)




    end

    def remove_items()





    end

    def prune_shopping_list(item_prices, item_budget)




    end
end
```

[15pts] OCaml Typing

1   (6pts)   Write an expression of each of the following types **without using type annotations**

  a. `float -> int -> int`

  b. `float * int list -> float list`

  c. `'a -> 'a list`

2   (6pts)   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 a b = a ^ b`

  b. `let x y z = y + z in`
     `let f i a = if (x i 3) = (a i 4) then "hello" else (a i 4)`

  c. `let f a b = (b @ b) :: a`

3 (3pts) Define a function **f** that when used in the following expression will calculate the sum of the list `[1; 2; 3; 4]`. The implementation and type of fold are given for reference, below.

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

fold: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
```

```
fold f 0.0 [1;2;3;4]
```

Write your implementation of **f** below:

[14pts] OCaml Execution

The code for map and fold are provided here for your reference:

```
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.

1 (2pts)

```
let rec f lst b =
    match lst with
    | [] -> true
    | h::t -> let sum = fold (fun a v -> a + v) 0 lst in
              (sum < b) && (f t b) in
                f [-5; 3; 1] 0
```

RESULT:

2 (3pts)

```
let foo fs lst =  fold (fun acc x -> (map x lst)::acc) [] fs in
    foo [(fun x -> x+1); (fun x -> x*2); (fun r -> r-1)] [1;2;3]
```

RESULT:

3 (2pts)

```
let x = 10 in let y = let x = 20 in x + x in x * y
```

RESULT:

4 (4pts)

```
type float_tree =
     Leaf
   | Node of string * float_tree * float_tree;;

let t1 = Node("r", Leaf, Node("o", Leaf, Leaf));;
let t2 = Leaf;;
let t3 = Node("w", Leaf, Leaf);;

let rec tfun t = match t with
    | Leaf -> ""
    | Node(s, l, r) -> tfun l ^ s ^ tfun r;;

map tfun [t1;t2;t3]
```

RESULT:

5 (3pts)

```
let rec f lst = match lst with
    | [] -> []
    | h1::h2::t -> (h1 + h2)::(f t)
    | h::t -> h::(f t);;

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

RESULT:

[16pts] OCaml Programming

The code for map and fold are provided here for your reference:

```
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   (8pts)   Define a function `is_sorted` that takes an int list and returns true if the list is sorted, and false if it is not sorted. You **may not** use the rec keyword in your solution. **Hint: you may find the fold/map functions above helpful**. Any solution that uses the rec keyword will receive no more than half credit for this question.

Example:

```
is_sorted [1; 1; 2; 3; 4; 5] = true
is_sorted [1; 5; 3] = false

let is_sorted lst =
```

Given a binary tree, where each node has a list of integer keys as shown in the figure below,
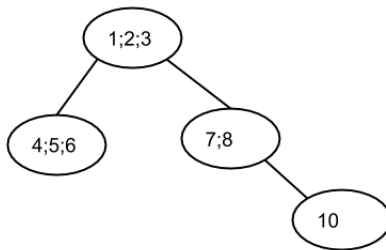
and the type of the tree is

```
type tree=
  | Leaf
  | Node of int list * tree *  tree
```

2   (8pts)   Write the function `sum` which returns the total sum of all keys in the tree.

```
   For example:  let t = Node ([1; 2; 3],
                                Node ([4; 5], Leaf, Leaf),
                                Node ([7; 8],
                                      Leaf,
                                      Node ([10], Leaf, Leaf)
                                )
                          )
 sum t = 40    (* 1+2+3+4+5+7+8+10 = 40 *)

let rec sum t =
```