CMSC330 Spring 2020 - 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.
- 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.

	Section	**Points**
1	Programming Language Concepts	10
2	Ruby Regular Expressions	11
3	Ruby Execution	15
4	Ruby Programming	18
5	OCaml Typing	15
6	OCaml Execution	14
7	OCaml Programming	17
	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: _____

1. [10 pts] Programming Language Concepts

Circle your answer. Each question is 1 point.

T F In statically typed languages, a type error will go unnoticed if the line containing the error is never executed.

T F Immutability is a key concept of functional languages.

T F Ruby code blocks are first-class; e.g., they can be passed into and returned from methods, and can be assigned to variables directly.

T F Closures are used to implement dynamic scoping.

T F OCaml uses a static type checking system.

T F If a programming language has type inference, then variable types are ignored until runtime.

T F The `map` function is an example of a higher order function.

T F In all languages with static type checking, the variable type must be explicitly declared.

T F The cons operator combines two lists together in OCaml.

T F Higher order functions refer to functions that take in other functions as arguments or return a function.

2. [11 pts] Ruby Regular Expressions

A. (2 points) Circle all the strings that match the regular expression:

```
/^[A-Z]*[a-z]+\s?.[0-3]+$/
```

Cmsc:330 CMSC.330 eastman .987 anwar :00001a

B. (5 points) Construct a regular expression that matches on email addresses:
- The username (the part before the '@' symbol) is a string of non-zero length consisting of only uppercase and lowercase letters, numbers and underscore.
- The domain (the part after the '@' symbol) should consist of only lowercase letters separated by periods (there will be at least one period, but will NOT start or end with a period).

Examples that should match:

"email@some.mail.site.url" "queenGG@gmail.com"
"reastman@cs.umd.edu" "aquaman@aol.com"

Examples that should **NOT** match:

"reastman@cs" "queenGG@gmail2.com"
"anwar@cs..umd.edu" "aquaman@aol.com.

Answer: **/^\w+@([a-z]+\.)+([a-z]+)$/**

C. (4 points) Write the output of the following Ruby code segment:

```
s = "Movie: Justice League, Time: 3:30PM, Seat: B4"
s =~ /^Movie: (\w+ )+\w*, Time: (\d+:\d*)(AM|PM), Seat: ([A-Z]{1}\d+)$/
puts "You are watching #{$1} at #{$2} in seat #{$3}"
```

Answer: **"You are watching Justice at 3:30 in seat PM"**

3. [15 pts] Ruby Execution

Write the output of the following Ruby code. Make sure to write your output exactly as ruby would print it (for example, pay attention to where there would be newlines). If there is an error, then write **ERROR**. If nil is printed write **"NIL"** and not the empty string.

A. (2 points)

```
arr = ["C", "M", "S", "C"]
arr[4] = 330
arr.each do |x|
      print x
end
```

Answer: CMSC330

B. (4 points)

```
def mystFunc(x, y)
      if (x.length <= y)
            puts yield x
      else
            puts yield x[0...y]
      end
end

mystFunc("CMSC330", 6) { |x| x.reverse }
mystFunc("330", 5) { |x|  x + x }
```

Answer: 33CSMC
330330

C. (2 points)

```
arr = ["C", "M", "S", "C"]
arr.sort
puts arr[3]
```

Answer: C

D. (4 points)

```
my_containers = [
   [1, "hello", true],
   { :hello => true, 2 => 1, 1 => 2},
   "hello",
   [[1,2,3], [4,5,6], [7,8,9]]
]

my_containers.each { |container|
    puts container[2]
}
```

Answer:

```
true
1
l
7
8
9
```

E. (3 points)

```
def do_stuff(s)
    s =~ /(\d+)/
    puts $1
end

r = "I'm having so much fun on this exam".scan(/\w+/)
puts r[2]
do_stuff("I love CMSC 330!")
do_stuff("351_132")
```

Answer:

```
having
330
351
```

4. [18 pts] Ruby Programming

You have been asked to rewrite the database system for a local babysitting company called the BabySitter's Club (BSC). It is your job to implement the `DayCareSystem` class.

Each baby will be represented by a single file. Here is an example of such a file:

```
Baby Name: Pao Fan
Baby Id: 115243231
Timeout: 10 minutes
```

Please read the description for all 4 functions first, as you may need to store information for use in other functions.

A. (3 points) **`initialize`** You will have to decide the proper data structure(s) for storing the data needed for the rest of the outlined functions. Initialize any necessary data structures in this method.

B. (7 points) **`loadBabyFromFile(file)`** This method will take in a filename and store the following information from the file (see below for an example file):
   a) Baby name: the name of the baby that the file describes. Contains only upper and lower case letters. If the name contains multiple words, they will be separated by exactly one space.
   b) An id is a string of digits that has at least a size of 1
   c) Timeout is a positive integer followed by the word "minutes"

   You must update your daycare system with the information you have read from the file. **You may assume the files are properly formatted**, so don't worry about what your program will do in the case of an invalid input text file. You **do not** have to worry about getting duplicate baby names.

C. (4 points) **`printBaby(name)`** This method will print out all of the stored information on the baby whose name is passed in as the argument `name`, in format shown below. Printing the baby defined in the example on the previous page should print

Pao Fan, 115243231, Timeout: 10

If no baby matches the provided name, print "not found: " followed by the provided name. For example, if a baby named "Billy" doesn't exist, calling `printBaby("Billy")` should print

not found: Billy

D. (4 points) **`graduateBaby(name)`** This method will "graduate" the baby out of the daycare, meaning that the baby will no longer be stored by the daycare system. All information on the baby should be removed. If no baby with the specified name exists, return `false`. Otherwise, return `true`.

You may find the following function useful:

`File.read(file_path)`: returns the entire file as a string (each line is separated by a newline character `\n`, including one after the last line).

```
class DayCareSystem
   def initialize()
      @system = {}




   end
   def loadBabyFromFile(file)
      lines = File.read(file).split("\n")
      if line[0] =~ /^Baby Name: (.+)$/
         name = $1
      end
      if line[1] =~ /^Baby Id: (.+)$/
         id = $1
      end
      if line[2] =~ /^Timeout: (.+) minutes$/
         timeout = $1
      end
      @system[name] = [id, timeout]









   end

   # class continued on next page
```

```
    def printBaby(name)
       baby = @system[name]
       if baby then
          puts "#{name}, #{baby[0]}, Timeout: #{baby[1]} minutes"
       else
          puts "not found: #{name}"
       end

    end
    def graduateBaby(name)
       if @system[name] then
          @system.delete(name)
          return true
       else
          return false
       end

    end
end
```

5. [15 pts] OCaml Typing

A) (6 points) Write an expression of the following type **without using type annotations** and **make sure any pattern matching is exhaustive** (that is, no input should cause a match failure).

a) `'a -> bool -> int -> 'a`

```
fun a b c -> if b || (c = 1) then a else a
```

b) `(float * int list * string) list`

```
[(1.0, [1], "hello")]
```

c) `float -> 'a -> ('a * int)`

```
fun a b -> if a = 1.0 then (b,1) else (b,1)
```

B) (9 points) Determine the types of the following expressions. Write **ERROR** if there is a type error.

a) `fun a b c -> if b = 0 then a else c b`

```
'a -> int -> (int -> 'a) -> 'a
```

b) `let pow x y = (x *. 2.0) + (2 * x * y) + (y * 2)`

ERROR

c) `let rec f x y z = if x then y::[z x] else (f x y z)`

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

6. [14 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 each of the following expressions. *If there is an error or exception, briefly explain what it is.*

A. (3 points)

```
map (fun (x,y) -> (y,x)) ((true,false)::[(false,true)])
```

Answer: `[(false,true); (true,false)]`

B. (2 points)

```
map (fun x -> x + 4) [1; 2; 3]
```

Answer: `[5; 6; 7]`

C. (3 points)

```
let rec f s n c =
     match s with
     | [] -> []
     | h :: t when (c n h) -> n :: h :: t
     | h :: t -> h :: f t n c

f [1;2;3;7;11;51] 17 (=)
```

Answer: [1; 2; 3; 7; 11; 51]

D. (3 points)

```
fold (fun acc x -> acc + x) [] [1; 2; 3]
```

Answer: Type Error - Initial Accumulator is wrong type

E. (3 points)

```
let g lst =
  (fold (fun acc (b, c) -> b + c) 0 lst)
in
  g [(1,2); (3,4); (5,6)]
```

Answer: 11

7. [17 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)
```

A. (3 points) Complete the implementation of the `insert` function that inserts an element `ele` into a sorted list `lst`. You may assume that `lst` is indeed sorted. You may use `map` or `fold` if you would like. You **may not** define any additional functions.

```
insert 5 [] = [5]
insert 0 [(-1); 1; 2] = [(-1); 0; 1; 2]
```

Complete the implementation by filling in the three blanks:

```
let rec insert ele lst =

     match lst with

     | [] -> [ele]

     | h::t -> if (ele < h) then

                    ele :: h :: t

               else

                    h :: (insert ele t)
```

B. (6 points) Complete the implementation of the `minimize` function that returns a tuple with two elements:
   - the last element of `lst` that minimizes the function `f`
   - the corresponding minimum value

   You **may not** declare any additional helper functions, except for the one given. You may not add the `rec` keyword to any function.

   Assume that the list is not empty.

```
minimize (fun x -> (-1) * x) [0; 1; 2] = (2, -2)
minimize (fun x -> List.length x) [[0; 1]; [4]] = ([4], 1)
minimize (fun x -> 0) [5; 1; 3] = (3, 0)
```

   Complete the function by filling in the arguments to `fold`:

```
let minimize f lst =
     match lst with
     | h :: _ -> fold
          (fun (a, b) x -> if (f x) <= b then
                               (x, f x)
                           else
                               (a, b))
          (h, f h)
          lst
```

C. (8 points) You are given the following type `'a tree` which defines a binary tree:

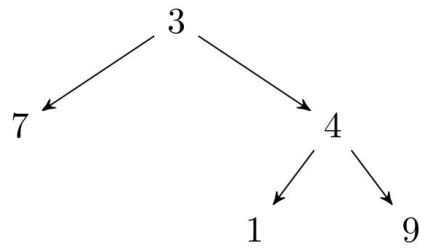
```
type 'a tree =
    Node of 'a * 'a tree * 'a tree
  | Nil
```

Define the following function `maximum` which returns the largest value in the tree. You may use `map` and `fold`, and you **may** define any number of recursive (or non-recursive) helper functions. You can assume that the tree will not be empty.

For example, this binary tree would be represented as follows:

```
let t = Node(3,
           Node(7, Nil, Nil),
           Node(4,
             Node(1, Nil, Nil),
             Node(9, Nil, Nil)))

maximum t = 9
```




```
let rec maximum t =
  match t with
  | Node(n, Nil, Nil) -> n
  | Node(n, l, Nil) -> max n (maximum l)
  | Node(n, Nil, r) -> max n (maximum r)
  | Node(n, l, r) -> max n (max (maximum l) (maximum r))
```