CMSC 330 Exam 1 Spring 2022 Solutions

Q2. PL Concepts

Q2.1. A single programming language can be compiled or interpreted, or both. T / F

Q2.2. In Ruby, the following is a type error that is caught before runtime: `"a" + 1` T / F

Q2.3. You cannot have a tuple of functions in OCaml, that is a tuple of the following signature:
`('a -> 'a) * ('b -> 'b)` T / F

Q2.4. `[7, 8, 9]` is valid in both Ruby and OCaml. T / F

Q2.5. In the given Ruby code, which of the following are true? Select all that apply.

```
a = [1, 2, 3]
b = [1, 2, 3]
(a.equal? b && a == b)
```

- `equal?` Returns `false` because `a` and `b` are not "structurally" equal
- `==` returns `true` because `a` and `b` are "structurally" equal
- The expression returns `false`
- The expression returns `true`

Q2.6. Consider the following Ocaml expression, where `e` is an arbitrary expression that evaluates to an `int` value.

```
let y = ref e in
let x = y in
let z = [|ref 1; ref 2; ref 3;|] in
let _ = z.(0) <- x in
let _ = y := 4 in
!(z.(0))
```

What is the output of the following code?

- 1
- 4
- The `int` value that the expression `e` evaluates to.
- Type error

Q2.7. Consider the following OCaml expression:

```
let x = 1 in
let f x y = 2*x + 3*y in
let g = f x in
let x = 4 in
g 3
```

Which of the following are true? Select all that apply.

- `g 3` returns `11`
- `g 3` returns 17
- The function `g` is equivalent to the function `(fun y -> 2 + 3 * y)`
- The type of `g` is `(int -> int -> int)`

Q3. Regular Expressions

Q3.1. Consider the following regex: `/[a-z]+.\d{3}\w*$/`

Which of the following strings will have a match (partial or exact) with the above regex? Select all that apply.

- cmsc330
- cmSC389F
- Cmsc320
- Cmsc4330

Q3.2. Write a regex that will exactly match email addresses with the following properties:

- Starts with one or more lowercase letters.
- Contains exactly 4 digits after the lowercase letters.
- Has an "@" after the 4 digits.
- Ends in "umd.edu" or "gmail.com" exactly.

Examples of valid emails:

```
bobby7857@gmail.com
sasha5465@umd.edu
guido1337@gmail.com
```

Examples of invalid emails:

```
mila8898@umd.com
123@gmail.com
martin12@gmail.com
sasha1@hotmail.com
```

Prompt:

```
/^[a-z]+[0-9]{4}@(umd\.edu|gmail\.com)$/
```

Q4. Ruby: Fill in the Blanks

Q4.1. Suppose that we have a function `word_sum` that takes in as input a hash, with string keys and integer values, and prints the sum of all values in the hash.

```
def word_sum(h)
  sum = 0
  h.keys.each { |k| sum += h[k] }
  puts sum
end
```

Examples:

```
word_sum({ "hello" => 1, "world" => 4, "CMSC" => 10}) # prints 15
word_sum({ "a" => 1, "lazy" => 4, "brown" => 5, "fox" => 3}) # prints 13
```

Q4.2. Fill in the blanks to implement a class `Dog` that keeps track of a dog's name and age and how many dogs were created.

```
class Dog
  @@count = 0
  def initialize(name, age)
    @name = name
```

```
    @age = age
    @@count += 1
  end

  def get_dog_years()
    @age * 7
  end

  def self.get_total_num_dogs
    @@count
  end
end
```

Examples:

```
dog = Dog.new("Nova", 2)
doggo = Dog.new("Chica", 4)
Dog.get_total_num_dogs
=> 2
doge = Dog.new("Astro", 10)
Dog.get_total_num_dogs
=> 3
```

Q4.3. Fill in the blanks to implement a function `get_avg_grade` that returns the average grade (as an integer) for a string formatted as: `Firstname, grade1, grade2`.

We will define a **valid** string as follows:

- A valid first name starts with an uppercase letter, followed by one or more lowercase letters.
- Commas should always be followed by a single space.
- Both valid grades are integers from 0 to 100. You don't have to worry about leading zeroes.

```
def get_avg_grade(data)
  if data =~ /^([A-Z][a-z]+), (\d{1,2}|100), (\d{1,2}|100)$/ then
    ($2.to_i + $3.to_i)/2
  else 0
  end
end
```

Note: The average grade must be rounded down and returned as an integer.

Examples:

```
get_avg_grade("Guido, 70, 80")
=> 75
get_avg_grade("Pranav, 85, 90")
=> 87
get_avg_grade("Yuzhu, 100, 100")
=> 100
```

Q5. Ruby: Coding

We're taking a trip back to the early 1900s! Back in the day, the Sears Roebuck Company was well known for their catalog kit houses (many of which are still around today!). We want to find out details about the different home models that the Sears Roebuck Company sold. You will be given a file called `sales.txt` which contains information about each home model that was sold.

If a model was purchased more than once (ideally by different people), that model and its selling price will appear in the file multiple times, once for each sale. Each line in `sales.txt` should be of the following format: `<Home Model Name>, <Selling Price>`

We will define a **valid** line as follows:

- The **Home Model Name** consists of any number of words, separated by spaces, where each word begins with an uppercase letter, followed by one or more lowercase letters.
- Commas should always be followed by a single space.
- The **Selling Price** will start with a dollar sign $ followed by one or more digits. It will be a whole dollar amount and have no commas, regardless of how large the dollar amount is.
- All invalid lines should be ignored.

Example of valid lines:

```
Fairfield, $2500
Cape Cod, $4500
Tudor, $0
Victorian, $5300
Cape Cod, $4000
Colonial, $3000
Fairfield, $2700
Cape Cod, $3200
```

Example of invalid lines:

```
Fairfield, $
Cape Cod , $10
, $100
Victorian, $1000.50
Cape Cod, 5400
Colonial2, $3000
```

You will have to implement four functions, described below:

1. `initialize(filename)`: Reads the file and parses the contents. Store the contents in any data structure you like, as long as these other functions work as described below.
2. `num_model_sold(model_name)`: Returns the number of kits of a particular model that were sold. If `model_name` does not exist, return `nil`.
3. `avg_selling_price(model_name)`: Returns the average amount of money that a particular model was sold for. All values should be rounded down to the nearest dollar. If `model_name` does not exist, return `nil`.
4. `total_sale_amount()`: Returns the total amount of money the Sears Roebuck Company made from sales of the kit homes. If no sales are made, return `0`.

Examples:

Suppose `sales.txt` contains all lines from the example of valid lines.

```
s = SearsRoebuck.new('sales.txt')
```

```
s.num_model_sold('Fairfield')
=> 2
s.avg_selling_price('Cape Cod')
=> 3900
s.total_sale_amount
=> 25200
```

Prompts:

```
class SearsRoebuck
  # Part 1
  def initialize(filename)
    # You may use this block to define your data structures
    @data = Hash.new
    # Process each line here!
    File.readlines(filename).each do |line|
      if line =~ /^([A-Z][a-z]+( [A-Z][a-z]+)*), \$(\d+)$/
        if @data[$1]
          @data[$1] << $3.to_i
        else
          @data[$1] = [$3.to_i]
        end
      end
    end
  end

  # Part 2
  def num_model_sold(model_name)
    if @data[model_name]
      @data[model_name].length
    else
      nil
    end
  end

  # Part 3
  def avg_selling_price(model_name)
    if @data[model_name]
      @data[model_name].sum / @data[model_name].length
    else
      nil
    end
  end

  # Part 4
  def total_sale_amount()
    total = 0
    @data.each {|k, v| total += @data[k].sum }
    total
  end
end
```

Q6. OCaml: Typing

- For all following three sub-questions, your expression's most general type must be the one given i.e. OCaml infers the expression's type exactly as the one given.
- You are **not allowed to use type annotations**.
- All pattern matching **must be exhaustive**.
- No other warnings should be raised.

Q6.1. Give an expression of the following type: `int -> string -> int option`

```
fun a b -> Some(a + (int_of_string b))
```

Q6.2. Give an expression of the following type: `(int -> bool) -> int -> bool list`

```
fun f x -> [f (x+1); true]
```

Q6.3. Give an expression of the following type: `'a -> ('a -> 'b) -> 'b -> bool list`

```
fun a b c -> [b a = c]
```

Q7. OCaml: What's the Input?

Q7.1. Consider the following function `foo`:

```
let foo g x =
  if g x = 5.0 then
    x
  else
    0
```

Suppose we have an arbitrary expression e such that:

```
foo e 5 = 5
foo e 3 = 0
```

What is the expression e such that it will produce the above outputs?

```
(fun x -> float_of_int x)
```

Q7.1. Consider the following function:

```
let op lst = fold_left (fun acc x -> match acc with
  | (i, l) when i mod 2 = 0 -> (i+1, l @ [x])
  | (i, l) -> (i+1, l)) (0, []) lst
```

Suppose we have a list `lst` such that:

```
op lst = (4, [1; 3])
```

What is the list `lst` such that it will produce the above output?

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

Q8. OCaml: Fill in the Blanks

Fill in the blanks to implement a function `string_filter` that takes in a list of strings as input and returns a new list consisting only of strings of length greater than or equal to the parameter `k` **in order**.

Notes: You may use the function `String.length` to determine the length of a string. No other functions from any external modules like `List` are allowed.

Examples:

```
string_filter ["330"; "is"; "the"; "best"] 3 = ["330"; "the"; "best"]
string_filter ["this"; "is"; "an"; "exam"] 4 = ["this"; "exam"]
string_filter [""; "a"; "aaaa"; "abc"] 1 = ["a"; "aaaa"; "abc"]
```

Prompt:

```
let string_filter lst k =
  fold_right (fun x a -> if String.length x >= k then x::a else a) lst []
```

Q9. OCaml: Coding

- You are not allowed to use any external modules such as `List`, etc.
- You are not allowed to use imperative OCaml
- For questions that require higher-order functions, we have provided you the following definitions:

```
let rec map f xs = match xs with
| [] -> []
| x :: xt -> (f x)::(map f xt)

let rec fold f a xs = match xs with
| [] -> a
| x :: xt -> fold f (f a x) xt

let rec fold_right f xs a = match xs with
| [] -> a
| x :: xt -> f x (fold_right f xt a)
```

Q9.1. Implement a **tail recursive** function repeat that takes in as input a string `s` and an integer `n` and returns a new string containing `s`, repeated `n` times. You may assume the given integer is greater than or equal to 0.

Examples:

```
repeat "hello" 0 = ""
repeat "cmsc330" 1 = "cmsc330"
repeat "buzz" 3 = "buzzbuzzbuzz"
repeat "na" 10 = "nanananananananananana"
```

Note: You are not allowed to use higher-order functions to implement the function.

```
let repeat s n =
  let rec aux s n acc =
    match n with
    | 0 -> acc
    | _ -> aux s (n-1) (s^acc)
  in aux s n ""
```

Q9.2. Consider the function `flatten` that takes as input an `int list list` and merges all inner `int lists` together into a single `int list`. No need to account for duplicates in the final list.

Examples:

```
flatten [[1; 2; 3]; [2; 4; 5]] = [1;2;3;2;4;5]
flatten [[]; [3;4;5]] = [3;4;5]
flatten [[]] = []
flatten [[1;1;1;2;3;4]; [4;3;4;2;5;6]] = [1;1;1;2;3;4;4;3;4;2;5;6]
```

Using higher-order functions, implement the function `flatten` below.

```
let flatten lst =
  fold_right (@) lst []
```