CMSC330 - Organization of Programming Languages
Spring 2023 - Exam 1

CMSC330 Course Staff
University of Maryland
Department of Computer Science

Name: Exam 1 Solutions

UID: _____

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination

Signature: _____

Ground Rules

- You may use anything on the accompanying reference sheet anywhere on this exam
- You may not use any built in functions if they are not on the reference sheet
- Please write legibly. If we cannot read your answer you will not receive credit
- You may not leave the room or hand in your exam within the last 10 minutes of the exam
- If anything is unclear, ask a proctor. If you are still confused, write down your assumptions in the margin

Question	Points
Q1	12
Q2	10
Q3	5
Q4	15
Q5	15
Q6	5
Q7	15
Q8	3
Q9	5
Q10	5
Total	90

Problem 1: Language Concepts

[Total 12 pts]

(a) True/False [6 pts]

Referential Transparency aims to minimize side effects Taken directly from notes and slides	**True**	**False**
Statically typed languages must also be explicitly typed Ocaml is both statically and implicitly typed	**True**	**False**
The features of a language influence how programs in that language are to be written Taken from the intro notes and was stated in lecture. We know Ocaml uses Referntial transparency which restricts how you may solve certain problems	**True**	**False**
Languages can only be declarative or imperative, not both Ocaml can be both, and from the notes: python can be both	**True**	**False**
Each language has it's own features, there is no overlap Java and C are both imperative	**True**	**False**
Dynamically Typed languages associate type with variables, not values Taken directly from slides. Also consider how Ocaml and Ruby differ in type inference	**True**	**False**

(b) Short Response: Languages [3 pts]

You have to design a translator that reads in text files and manipulates strings. Would you want to use C, Ruby, or Ocaml here? Give one advantage that language gives and one disadvantage that one of the other languages have.

Answers may vary: pick one of the three languages and give a valid and appropriate advantage and disadvantage.

(c) Short response: Side Effects [3 pts]

What are side effects and what is a problem they cause?

Side effects (in the context of this class) are when variables outside parameters or scope of a function are modified.

They can cause programs to have unexpected behavior or variable output (running the same function will not return the same thing each time).

Problem 2: Regular Expressions

[Total 10 pts]

(a) Which of the following strings are an exact match of the regular expression `/^[PC][A-Z]{3}.[0-9]+$/` [2 pts]

CMSC389T `CMSC330` `PHIL.410` `CHEM.310`

(b) Capture Groups [3 pts]

Write a regular expression that takes in a string of lowercase characters and captures the first and last vowel.

`/^[^aeiou]*([aeiou])[a-z]*([aeiou])[^aeiou]*$/`

Explanation: the first vowel can be found after looking at all non-vowels: `[^aeiou]*([aeiou])`. We then can see any lowercase letter, vowel or not until we see a vowel potentially followed by non-vowels: `[a-z]*([aeiou])[^aeiou]*`

[5 pts] (c) Street Addresses

Write a regex that will match only on valid addresses. A valid street address is comprised of the following:

NUMBER NAME, CITY, STATE

- A NUMBER consists of one to five digits in a row
- A NAME consists of 1 or more WORDs where a WORD starts with a capital letter and is followed by zero or more lowercase letters. WORDs are seperated by a single space character.
- A CITY consists of one or more lowercase alphanumeric characters. There are no whitespace characters in CITY.
- A STATE consists of exactly 2 capital letters

Examples of valid addresses

- 123 Street Drive, mycity, ST
- 0 Streety Mcstreetface1, internetpolls, HA
- 54321 A, b, CD

Examples of invalid addresses:

- 123456 Too Many Digits, city, ST
- 0 no cap, Wrongcap, HA,
- 12 No Commas And toomanystateletters OOF

```
/[0-9]{1,5}( [A-Z][a-z]*)+, [a-z0-9]+, [A-Z]{2}/
```

Explanation: A number of 1 to 5 digits is given by `/[0-9]{1,5}/`

A word starts with a capital letter and has zero or more lowercase: `[A-Z][a-z]*`. Then one or more words separated by spaces `( [A-Z][a-z]*)+`.

A city is one or more alphanumeric characters with no whitespace: `[a-z0-9]`

A state is two captial letters: `[A-Z]{2}`

[Total 5 pts]

Problem 3: Project 1

Recall Project 1b: Describe your data structure(s) in terms of how you looked up a translation (if you do not have a working implementation, write how you would do it)

Answers may vary: make sure you explain your data structure(s) well enough to give context for the answer below.

Suppose there was a third file you had to read in: `tenses.txt` which held lines like

```
run, English:  present:running, past:ran, future:will run
```

Describe how your data structure(s) would change (or if you would need to make a new one) to incorporate a new method `change_tense(sentence, tense)` where you had to convert words with a part of speech VER in `sentence` to `tense`. For example, assuming `"ran, VER"` and `"he, PRO"`, `change_tense("he ran", "future") -> "he will run"`.

Answers may vary: correctly describe how the previously mentioned data structure would include another structure for tenses or describe the creation of another data structure to use for tenses.

Problem 4: Ruby Debugging [Total 15 pts]

Given the following Code:

There are at least 3 unique types of bugs in the code. Identify 3 unique types, write the line number, explain why its that type, then how to fix it. The program should print "Hello World3". There is one syntax, one type, and one logic bug.

```
1  def mystery(a,b,c)
2      input = c + b
3      yield input
4  end
5
6  puts mystery(Proc.new{|x| x + "Hello "}, "World", 3)
```

(a) Bug 1 [5 pts]

Logic error: In line 6, `x + hello` or in line 2, `c + b` will result in the string being in the wrong order.

Fix: `"hello" + x, b + c.`

(b) Bug 2 [5 pts]

Type Error: In line 6, `3 + "World"`, you cannot concat an int and a string together.

Fix: `3.to_s + "World"`

(c) Bug 3 [5 pts]

Syntax error: In line 3 you cannot yield if no codeblock is given

Fix: `a.call(input)`

Problem 5: Ocaml Typing [Total 15 pts]

Given the following Type, write an expression that matches that type. You may not use type annotations and all pattern matching must be exhaustive.

(a) `float -> float -> int list` [3 pts]

```
fun x -> fun y -> if (x +.  3) = y then [1] else [2]
```

Explanation: we need a function that takes in 2 arguments: `fun x -> fun y`

We need x to be a float so let's apply a float operation: `x +.  3`. We know that y also has to be a float so lets compare them: `(x +.  3) = y`

Now we just need to return an int list so let's wrap this all in a if expression.

[3 pts] (b) `float * int -> int * float -> int`

```
fun x -> fun y -> let (a,b) = x in let (c,d) = y in let e = a +.  d in b + c
```

Explanation: we need a function that takes in 2 arguments: `fun x-> fun y`

We need x and y to be a tuple so let's match them in a let expression: `let (a,b) = x in let (c,d) = y`

We need to make a and d floats so let's do float addition: `let e = a +.  d`

We need to return an int and make b and c ints, so let's just return the result of adding them: `b + c`

Given the expression, write down it's type:

[3 pts] (c) `fun h g z -> (h ((g 3) +.  4.0))::z`

```
float -> 'a -> int -> float -> 'a list -> 'a list
```

Explanation: Taken directly from the review done in lecture. `z` must be a list, being the right side operator for the cons operator.

g takes in 3 which is an int, and returns a float, since the return value uses the `+.` operator.

h takes in the result of a float operation and cons' the result onto a list.

Since we are given no information on the list, we give it a `'a` type.

[3 pts] (d) `fun a b c d -> (a b) < (c d)`

```
('a -> 'b) -> 'a -> ('c -> 'b) -> 'c -> bool
```

`Explanation:` we know a and b are both functions that take in a single parameter and return the same result, since the results can be compared.

We know a takes in the variable b and c takes in the variable d, but are given no more information about the types so we give them their respective `'a`, and `'c` types.

[3 pts] (e) `fun a b c -> if (a c) then (c + 3)::[] else b`

```
int -> bool -> int list -> int -> int list
```

Explanation: we know c must be an int, since we are `+ 3` to it.

We know a is a function that takes in c (which we now know is an int) and returns a bool (given it's position in the if expression)

We know that an int list is the return type of the true branch, making b the same type in the else branch.

Problem 6: Project 2

[Total 5 pts]

(a) Tree Fold

[5 pts]

Here is a `fold_tree` implementation.

```
let rec fold_tree f b t =
  match t with
  | Leaf           -> b
  | Node (l, v, r) -> f (fold_tree f b l) [v] (fold_tree f b r)
```

Suppose we write a `pre_order` function that returns the pre_order traversal using tree fold. How would the result of calling `pre_order` on a tree change if we keep `f` the same but change `fold_tree` to the following?

```
let rec fold_tree f b t =
  match t with
  | Leaf           -> b
  | Node (l, v, r) -> f (fold_tree f b r) (fold_tree f b l) [v]
```

This changes it from a pre-order to a post-order traversal.

Explanation: We know that if preorder takes root, left, right, meaning the function f, must do something like `[v]@left@right`.

Thus, we are just changing the order of the parameters, then we are getting `left, right, root`.

Problem 7: Ocaml Debugging

[Total 15 pts]

Consider the following code. `square` is a function that takes in a `linkedlst` and square each value in the list.

```
1  type linkedlst = Null|Node of int * linkedlst;;
2
3  let rec square lst = match lst with
4  Null -> []
5  | Node(x,t) -> Node(x^2, t)
```

There are at least 3 unique types of bugs in the code. Identify 3 unique types, write the line number, explain why its that type, then how to fix it. There is one syntax, one type, and one logic bug.

(a) Bug 1

[5 pts]

Logic error: In line 5, this just modifies the root node, no recursive call to modify the rest of the tree

Fix: `Node(x^2,square(t))`

(b) Bug 2

[5 pts]

Syntax/Type error: In line 5, `x^2` is for strings and this will not square numbers

Fix: `Node(x*x)`

[5 pts] (c) Bug 3

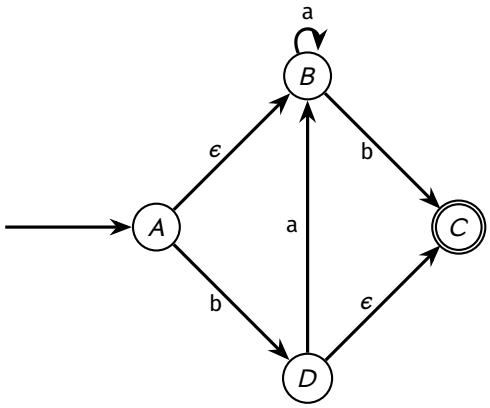
Syntax/Type: In line 4, returning two different types in each branch of the match expression.

Fix: `Null -> Null`

[Total 3 pts]

Problem 8: Finite State Machines

Given the following finite state machine, which of the strings would be accepted?



aa aaaab baaab b

Explanation: This is an NFA. If we can show at least one path out of all possible paths end in state C, then we say the machine accepts the string.
"aa" has the following possible paths -> "A -ϵ-> B -a-> B -a-> B" and "A -a-> garbage". Either way, it ends in a non accepting state.
"aaaab" has the following path: "A -ϵ-> B -a-> B -a-> B -a-> B -a-> B -b-> C"
"baab" has the following path: "A -b-> D -a-> B -a-> B -a-> B -b-> C"
"b" has the following path: "A -b-> D -ϵ-> C"

Problem 9: Ruby Coding

[Total 5 pts]

Write a program that does the following:

- read in a file where each line is a string of 1 or more alphabetic symbols followed by 1 or more numeric symbols. You may assume that all lines will be valid. The following are example strings in the file
  - alphabet1234
  - a2
  - COULDBECAPITAL09343
- calculate how many digits are in each number
- add up the sum of these values

```
1  def extractNumber(line)
2      ____Blank_1____
3  end
4
5  def numDigits(number)
6      ____Blank_2____
7  end
8
9  def sumOfArr(acc, item)
10     ___Blank_3____
11 end
12
13 file = File.open("myfile.txt")
14 numbers = file.readlines.map{|line| extractNumber(line)}
15 num_of_digits = numbers.map{|number| numDigits(number)}
16 puts num_of_digits.reduce(0){|acc, item|sumOfArr(acc, item)} # reduce is just like fold
```

(a) Blank 1 [2 pts]

```
/[a-zA-Z]*(\d*)/ =~ line

$1

# returning anything (like $1.to_i) works as long as it is consistent to blank 2
```

(b) Blank 2 [2 pts]

```
number.length
```
or
```
num = number.to_i

count = 0

while num != 0

   count + 1

   num/10

end

count

# Again, anything could work as long as it is consistent with blank 3 and blank 1
```

[1 pts] (c) Blank 3

```
acc + item

# Again, anything could work as long as it is consistent with blank 2
```

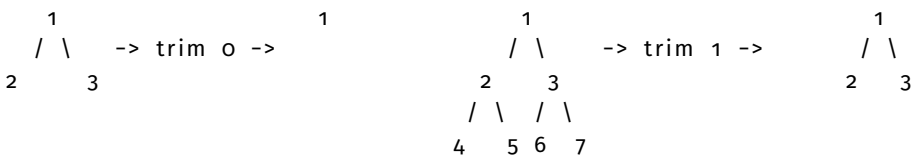
[Total 5 pts]

Problem 10: Ocaml Coding

Write a recursive function called `trim` that takes in a perfect binary tree tree and an integer n, and trims off nodes at the bottom of the tree such that the depth of the returned tree is at most n. You may use the following Tree Variant:

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

Examples:




```
let rec trim tree n =
  match tree with
  |Leaf -> Leaf
  |Node(v,l,r) -> if n < 0 then Leaf else Node(v, trim l (n-1), trim r (n-1))
```

Ruby

```
# Arrays
arr = []
arr = Array.new # makes an empty array
arr = Array.new(1) # makes an array of size one with default value nil
arr = Array.new(1,"A") # makes an array of size 1 with default value "A"

# Calls the given block once for each element in self, passing that element
# as a parameter.
Returns the array itself.
arr.each{|item| block}
for item in arr

# Returns a new array containing all elements of the array for which the
# given block returns a
true value
arr.select{|item| block}

# Hashes
h = {} #makes an empty hash
h = Hash.new # makes an empty hash
h = Hash.new(3) # makes the default value for any key 3

#Calls block once for each key in the hash, passing the key-value pair
# as parameters.
h.each{|key, value| block}
for k,v in h
for kvpair in h

# Map and Reduce(fold)

# Map: Calls the given block once for each element in self, passing that
# element as a paramter. Will return a new array containing all the results
# of the block for each element
arr.map{|item| block}
# equivalent in ocaml: map (fun item -> block) arr

# Reduce: Combines all elements of enum by applying a binary operation,
# specified by a block or a symbol that names a method or operator.

arr.reduce(initial){|acc, item| block}
# equivalent in ocaml: to fold (fun acc item -> block) initial arr
```

Regular Expression Documentation:

Creating a Regex:

/*pat*/	`/hay/ =~ "haystack"`
%r{*pat*}	`%r{hay} =~ "haystack"`

`[]`	range specification (e.g., [a-z] means a letter in the range a to z)	`{m,n}`	at least m and at most n repetitions of the preceding
`\w`	letter or digit; same as [0-9A-Za-z]	`{m,}`	at least m repetitions of the preceding
`\W`	neither letter or digit	`(a|b)`	a or b
`\s`	space character; same as [\t\n\r\f]	`(...)`	grouping; capture everything enclosed
`\S`	non-space character	`^`	Start of line
`\d`	digit character; same as [0-9]	`$`	End of line
`\D`	non-digit character	`[^abc]`	Any single character except: a, b, or c
`*`	zero or more repetitions of the preceding	`[abc]`	A single character of: a, b, or c
`+`	one or more repetitions of the preceding	`[a-z]`	Any single character in the range a-z
`?`	at most one repetition of the preceding; same as {0,1}	`a{3}`	Exactly 3 of a

Matching a Pattern:

/regexp/ =~ string	`/hay/ =~ "haystack"   #=> 0` `/needle/.match('haystack') #=> nil`

/regexp/.match(string)	`/y/.match('haystack') #=> #<MatchData "y">`
	`/I(n)ves(ti)ga\2ons/.match("Investigations")`
	`#=> #<MatchData "Investigations" 1:"n" 2:"ti">`

case... when...end

`case`	Starts a case statement definition. Take the variable you are going to work with.
`when`	Every condition that can be matched is one when statements.
`else`	If nothing matches then do this. Optional.

Ruby Case & Ranges

```
case capacity
when 0
  "You ran out of gas."
when 1..20
  "The tank is almost empty. Quickly, find a gas station!"
when 21..70
  "You should be ok for now."
when 71..100
  "The tank is almost full."
else
  "Error: capacity has an invalid value (#{capacity})"
end
```

Grammar for Regular Expressions:

$$
\begin{array}{rcl}
R & := & \emptyset \\
 & | & \sigma \\
 & | & \epsilon \\
 & | & R_1 R_2 \\
 & | & R_1 | R_2 \\
 & | & {R_1}^*
\end{array}
$$

Here are the type definitions and signatures for some useful OCaml functions:

```
map : ('a -> 'b) -> 'a list -> 'b list
let rec map f l = match l with
    [] -> []
    |h::t -> (f h)::(map f t)

fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
let rec fold_left f a l = match l with
    [] -> a
    |h::t -> fold_left f (f a h) t

fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
let rec fold_right f l a = match l with
    [] -> a
    |h::t -> f h (fold_right f t a)

cons: 'a -> 'a list -> 'a list
append: 'a list -> 'a list -> 'a list
```