

CMSC330 - Organization of Programming Languages Spring 2023 - Exam 1

CMSC330 Course Staff
University of Maryland
Department of Computer Science

Name: _____

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	True	False
Statically typed languages must also be explicitly typed	True	False
The features of a language influence how programs in that language are to be written	True	False
Languages can only be declarative or imperative, not both	True	False
Each language has it's own features, there is no overlap	True	False
Dynamically Typed languages associate type with variables, not values	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.

(c) Short response: Side Effects

[3 pts]

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

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.

[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

[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)

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".

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]

(b) Bug 2

[5 pts]

(c) Bug 3

[5 pts]

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]

(b) float * int -> int * float -> int

[3 pts]

Given the expression, write down it's type:

[3 pts]

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

[3 pts]

(d) `fun a b c d -> (a b) < (c d)`

[3 pts]

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

[Total 5 pts]

Problem 6: Project 2

[5 pts]

(a) Tree Fold

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]
```

Problem 7: Ocaml Debugging

[Total 15 pts]

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

```
1 type linkedList = Null | Node of int * linkedList;;
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]

(b) Bug 2

[5 pts]

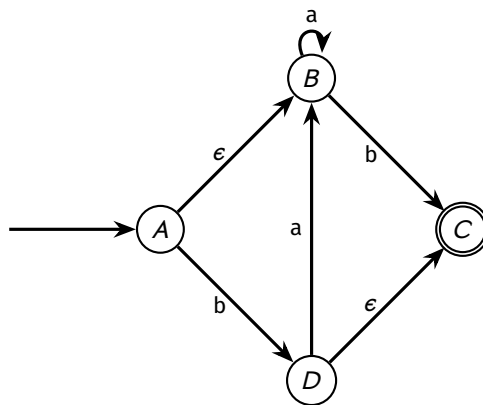
(c) Bug 3

[5 pts]

Problem 8: Finite State Machines

[Total 3 pts]

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



aa aaaab baaab b

[Total 5 pts]

Problem 9: Ruby Coding

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
 - COULDBECAPITALo9343
- 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
```

[2 pts]

(a) Blank 1

[2 pts]

(b) Blank 2

[1 pts]

(c) Blank 3

Problem 10: Ocaml Coding

[Total 5 pts]

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:

```
      1          1
     / \      / \
    2   3    2   3
           / \ / \
          4  5 6  7

-> trim 0 ->      1
                / \
               2   3

-> trim 1 ->      1
                / \
               2   3
```

```
let rec trim tree n =
```


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 parameter. 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:

<code>/pat/</code>	<code>/hay/ =~ "haystack"</code>
<code>%r{pat}</code>	<code>%r{hay} =~ "haystack"</code>

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

Matching a Pattern:

<code>/regexp/ =~ string</code>	<code>/hay/ =~ "haystack" #=> 0</code> <code>/needle/.match('haystack') #=> nil</code>
<code>/regexp/.match(string)</code>	<code>/y/.match('haystack') #=> #<MatchData "y"></code> <code>/I(n)ves(ti)ga\2ons/.match("Investigations")</code> <code>#=> #<MatchData "Investigations" 1:"n" 2:"ti"></code>

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:

```
R := ∅
   | σ
   | ε
   | R1R2
   | R1|R2
   | R1*
```

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
```