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  

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>12</td>
</tr>
<tr>
<td>Q2</td>
<td>10</td>
</tr>
<tr>
<td>Q3</td>
<td>5</td>
</tr>
<tr>
<td>Q4</td>
<td>15</td>
</tr>
<tr>
<td>Q5</td>
<td>15</td>
</tr>
<tr>
<td>Q6</td>
<td>5</td>
</tr>
<tr>
<td>Q7</td>
<td>15</td>
</tr>
<tr>
<td>Q8</td>
<td>3</td>
</tr>
<tr>
<td>Q9</td>
<td>5</td>
</tr>
<tr>
<td>Q10</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>90</td>
</tr>
</tbody>
</table>
Problem 1: Language Concepts

(a) True/False

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 its own features, there is no overlap
True False

Dynamically Typed languages associate type with variables, not values
True False

(b) Short Response: Languages

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

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

Problem 2: Regular Expressions

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

CMSC389T CMSC330 PHIL.410 CHEM.310

(b) Capture Groups

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 separated 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

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 it's that type, then how to fix it. The program should print "Hello World3". There is one syntax, one type, and one logic bug.

```ruby
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
(b) Bug 2
(c) Bug 3

Problem 5: Ocaml Typing

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

(b) float * int -> int * float -> int
Given the expression, write down its type:

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

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

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

Problem 6: Project 2

(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

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

```ocaml
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

(b) Bug 2

(c) Bug 3

Problem 8: Finite State Machines

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

```
a

A B C D

a
b
e
```

aa aaaaab baaab b
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 def numDigits(number)
  5 ____Blank_2____
  6 end

7 def sumOfArr(acc, item)
  8 ___Blank_3____
  9 end

10 file = File.open("myfile.txt")
11 numbers = file.readlines.map{|line| extractNumber(line)}
12 num_of_digits = numbers.map{|number| numDigits(number)}
13 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

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:

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

Examples:

```
1               1               1
/ \
   -> trim 0 ->          / \    -> trim 1 ->    / \
2   3             2   3             2   3
   / \               / \          / \
   4   5 6 7
```

```ocaml
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:

<table>
<thead>
<tr>
<th>Regex</th>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/pat/</code></td>
<td><code>/hay/ =~ &quot;haystack&quot;</code></td>
<td></td>
</tr>
<tr>
<td><code>%r{pat}</code></td>
<td><code>%r{hay} =~ &quot;haystack&quot;</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>range specification (e.g., [a-z] means a letter in the range a to z)</td>
<td><code>[]</code></td>
</tr>
<tr>
<td>\w</td>
<td>letter or digit; same as [0-9A-Za-z]</td>
<td><code>{m,}</code></td>
</tr>
<tr>
<td>\W</td>
<td>neither letter or digit</td>
<td>`(a</td>
</tr>
<tr>
<td>\s</td>
<td>space character; same as [ \t\n\r\f]</td>
<td><code>(\.)</code></td>
</tr>
<tr>
<td>\S</td>
<td>non-space character</td>
<td><code>^</code></td>
</tr>
<tr>
<td>\d</td>
<td>digit character; same as [0-9]</td>
<td><code>$</code></td>
</tr>
<tr>
<td>\D</td>
<td>non-digit character</td>
<td><code>[^abc]</code></td>
</tr>
<tr>
<td>*</td>
<td>zero or more repetitions of the preceding</td>
<td><code>[*\d\D]</code></td>
</tr>
<tr>
<td>+</td>
<td>one or more repetitions of the preceding</td>
<td><code>[+\w\s\S\d\D]</code></td>
</tr>
<tr>
<td>?</td>
<td>at most one repetition of the preceding; same as {0,1}</td>
<td><code>a{3}</code></td>
</tr>
</tbody>
</table>

Matching a Pattern:

<table>
<thead>
<tr>
<th>Regex</th>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/regexp/ =~ string</td>
<td>/hay/ =~ &quot;haystack&quot; =&gt; 0 /needle/.match('haystack') =&gt; nil</td>
<td></td>
</tr>
<tr>
<td>/regexp/.match(string)</td>
<td>/y/.match('haystack') =&gt; #&lt;MatchData &quot;y&quot;&gt; /I(n)ves(ti)ga\2ons/.match(&quot;Investigations&quot;) =&gt; #&lt;MatchData &quot;Investigations&quot; 1:&quot;n&quot; 2:&quot;ti&quot;&gt;</td>
<td></td>
</tr>
</tbody>
</table>
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:

\[ R \quad ::= \quad \emptyset \\
\quad \mid \quad \sigma \\
\quad \mid \quad \epsilon \\
\quad \mid \quad R_1 R_2 \\
\quad \mid \quad R_1 | R_2 \\
\quad \mid \quad R_1^* \]

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

```ocaml
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
```