CS131 Spring 23 - Midterm #1

May 4th, 2023

Full Name (**LAST**, FIRST): _____

Student ID: _____

Signature: _____

Problem #1	/10
Problem #2	/5
Problem #3	/16
Problem #4	/10
Problem #5	/5
Problem #6	/10
Problem #7	/9
Problem #8	/12
Total	**/77**

Please practice academic integrity - don't cheat.

1. [10 points-2 each] For each of the items below, write a Haskell type signature for the expression. If the item is a function then use the standard haskell type notation for a function. If the item is an expression, write the type of the expression (e.g., [Int]). For any answers that use one or more type variables *where concrete types can't be inferred*, use "t1", "t2", etc. for your type variable names.

a.
```
foo bar = map (\x -> 2 * x) bar
```
b.
```
y = \x -> x ++ ["carey"]
```
c.
```
m = [((\a -> length a < 7) x,take 3 y) |
      x <- ["matt","ashwin","siddarth","ruining"],
      y <- ["monday","tuesday","wednesday","thursday","friday"]]
```
d.
```
z = foldl (\x y -> x ++ y)
```
e.
```
foo bar = []
```

Answers:

a. _____

b. _____

c. _____

d. _____

e. _____

2. [5 points] Suppose a language does not support closures. Is it possible for the language to still support currying and partial function application? If so, how would this work; if not, explain why not. Note: Limit your answer to five or fewer sentences, and avoid adding superfluous, incorrect answers as this will result in a point deduction.

3. [16 points-2/2/2/10] In this question, you will be implementing *directed graph* algebraic data types and algorithms in Haskell. Recall that a graph is composed of nodes (aka vertices) that hold values, and edges which connect nodes to other nodes. In a directed graph, edges are unidirectional, so an outgoing edge from node A to node B does NOT imply that B also has a direct connection to A. You may assume that all graphs have at least one node.

a. Show the Haskell definition for an algebraic data type called "Graph" that has a single “Node” variant. Each Node must have one *Integer* value, and a list of zero or more adjacent nodes that can be reached from the current node.

b. Using your Graph ADT, write down a Haskell expression that represents a graph with two nodes:

1. A node with the value 5 with no outgoing edges
2. A node with the value 42, with an outgoing edge to the previous node

c. Carey has designed his own graph ADT, and written a Haskell function that adds a node to the “front” of an existing graph g that is passed in as a parameter:

```
add_to_front g = Node 0 [g]
```

In Haskell, creating new data structures incurs cost. Assuming there are N nodes in the existing graph *g*, what is the time complexity of Carey's function (i.e., the big-O)? Why?

d. Write a function called *sum_of_graph* that takes in one argument (a Graph) and returns the sum of all nodes in the graph. Your function must include a type annotation for full credit. You may assume that the passed-in graph is guaranteed to have no cycles. You may have helper function(s). *Your solution MUST be shorter than 8 lines long.*

4. [10 points] Write a Haskell function called *get_every_nth* that takes in an Integer n and a list of any type that returns a sublist of every n^{th} element. For example:

```
get_every_nth 2 ["hi","what's up","hello"] should return ["what's up"]
get_every_nth 5 [31 .. 48] should return [35, 40, 45]
```

Your function must have a type signature and must use either filter or foldl. It must be less than 8 lines long.

Hint: If you use fold, your accumulator doesn't need to be the same type as your returned value or your input, and a tuple might be useful!

5. [5 points] Write a Haskell comprehension that generates an infinite list of functions, each of which takes a single argument x, such that the k^{th} function returns x^k. You may assume that k starts at 1 for the first generated function, 2 for the second function, etc. For example:

inf_list = [your comprehension here]
head inf_list 9 → returns 9 since 9^1 is 9
head (tail inf_list) 9 → returns 81 since 9^2 is 81

6. [10 points-5/5] Consider the following Python program:

```
class Comedian:
  def __init__(self, joke):
    self.__joke = joke

  def change_joke(self, joke):
    self.__joke = joke

  def get_joke(self):
    return self.__joke

def process(c):
 # line A
 c[1] = Comedian("joke3")
 c.append(Comedian("joke4"))
 c = c + [Comedian("joke5")]
 c[0].change_joke("joke6")

def main():
 c1 = Comedian("joke1")
 c2 = Comedian("joke2")
 com = [c1,c2]
 process(com)
 c1 = Comedian("joke7")
 for c in com:
  print(c.get_joke())
```

a. Assuming we run the main function, what will this program print out?

b. Assuming we removed the comment on line A and replaced it with:

```
c = copy.copy(c)        # initiate a shallow copy
```

If we run the main function, what would the program print?

7. [9 points-3/3/3] We know that int and float are usually not subtypes of each other. Assume you have a programming language where you can represent integers and floating-point values with infinite precision (call the data types Integer and Float). In this language, the operations on Integer are +, -, *, / and % and the operations on Float are +, -, *, and /.

a. In this language, will Integer be a subtype of Float? Why or why not?

b. In this language, will Float be a subtype of Integer? Why or why not?

c. If you discovered that the Float type also supports an additional operator, exponentiation, what effect would that have on your answers to a and b? Why?

8. [12 points-3/3/3/3] Siddarth has created a compiler for a new language he's invented. He gives you the following code written in the language, with line numbers added for clarity:

```
00  func mystery(x) {
01    y = x;
02    print(f"{y}");
03    y = 3.5;
04    print(f"{y}");
05    return y;
06  }
07
08  q = mystery(10);
09  print(f"{q}");
```

Siddarth tells you that the program above outputs the following:

```
10
3
3
```

a. What can you say about the type system of the language? Is it statically typed or dynamically typed? Explain your reasoning.

b. Is either casting or conversion being performed in this code? If so, which technique(s) are being used on what line(s)? If you do identify any casts/conversions, explain for each if they are narrowing or widening.

c. Assuming the language used the opposite type system as the one that you answered in part a, what would the output of the program be?

d. Ruining likes the language so much that she built her own compiler for the language, but with some changes to the typing system. Siddarth wants to determine what typing system Ruining chose for her updated language, so he changed line 3 above to:

```
y = “3”;
```

and found that the program still runs and produces the same output. What can we say about the typing system for Ruining's language? Is it static or dynamic? Or is it impossible to tell? Why?