CS131 Fall '23 Midterm

Nov 7th, 2023

Student ID #: _____

Full Name (First, Last): _____

Practice Academic Integrity - Don't cheat!
(There are multiple versions of the exam, so copying from a neighbor will only get you caught - trust us!)

Problem #1: Object Reference Madness	/10
Problem #2: Algebraic Data Type Silliness	/15
Problem #3: Map, Filter, Repeat	/12
Problem #4: I'm Partial to Curry(ing)	/12
Problem #5: Eggert's Scoping and Typing	/12
Problem #6: This 'n' That	/12
Total	**/73**

1. Object Reference Madness (10 points)

In this problem, you must figure out what the following Python script prints when you run it:

```
class Potato:
    def __init__(self, x):
        self.weight = x
        self.bites = []

    def bitten_by(self, name):
        self.weight -= 1
        self.bites = self.bites + [name]
        return self.bites

def foo(potat):
    names = potat.bitten_by("Andrey")
    names = names + ["Justin"]

def main():
    p1 = Potato(5)

    names = p1.bitten_by("Bonnie")

    names.append("Carey")
    print(p1.bites) # Line A

    p1.bitten_by("Brian")
    print(p1.bites) # Line B
    print(names)    # Line C

    foo(p1)
    print(p1.bites) # Line D
    print(names)    # Line E

if __name__ == "__main__":
    main()
```

WRITE YOUR ANSWER ON THE NEXT PAGE

Answer Problem 1 here:

a. (2 points) What does Line A print?

b. (2 points) What does Line B print?

c. (2 points) What does Line C print?

d. (2 points) What does Line D print?

e. (2 points) What does Line E print?

2. Algebraic Data Type Silliness (15 points)

Given the following linked-list algebraic data type in Haskell:

```
data List = Node Int List | Nil
```

a. (2 points) Show the Haskell expression to create a linked list comprised of Nodes and Nil that holds the values 1,2,2,3,3

list_with_dups =

b. (10 points) Write a function called *dup_rem* that removes consecutive duplicate values from a List (like the one you defined above) and returns a new List that contains only the non-duplicated items. You may assume that the List only holds **positive integers** (> 0) in **ascending order** (e.g., 1,1,2,2,2,3,4,4), and that the List may be empty. So using *dup_rem* on a List containing the values 1,1,2,2,2,3,4,4 should produce an output list 1,2,3,4. Here's how it might be used:

```
-- outputs a List containing 3 nodes with values 1, 2 and 3.
list_wo_dups = dup_rem list_with_dups
```

Here are the requirements for your function:

- **It must be less than 15 lines long**
- **It must include a type signature**
- It may use a helper function

Hint: You may find a tuple helpful.

WRITE YOUR ANSWER ON THE NEXT PAGE

b. Answer Problem 2.b here:

c. (3 points) Assuming you pass in a linked list that contains the following values 10,10, 20, 30, 40, 50, 60 to your *dup_rem* function, how many new Node values are created (not Nil values, but just Node values) during the execution of the function. **For full credit, explain why.**

3. Map, Filter, Repeat! (12 points)

a. (2 points) Write a Haskell function named *extract2nd* that accepts a list of tuples as its only argument and returns a list of the second element from each tuple in the same order. **You must use *map*, *filter* or *foldl/foldr*, and provide the function type signature** for full credit.

For example:

```
extract2nd [('a', 1), ('b', 2), ('c', 4)]
```

returns: `[1, 2, 4]`

Write your answer here:

b. (2 points) Write a Haskell function named *filterBy1st* that accepts a list of tuples and a value of the same type as the first element in the tuples. This function must return a new list that removes all tuples from the input list where the first element matches the value. **You must use *map*, *filter* or *foldl/foldr*.** You do NOT need to include the type signature for full credit.

For example:

```
filterBy1st [(11,"a"), (22,"b"), (33,"c"), (22,"d")] 22
```

returns: `[(11,"a"), (33,"c")]`

c. (4 points) Write a Haskell function named *removeElemAtIndex* that accepts a list of integers and an index (zero-indexed) as its parameters. The function must return a new list with the element at the given index removed. **Your function must use the *extract2nd* and *filterBy1st* functions.** You do NOT need to include the type signature for full credit.

For example:

```
removeElemAtIndex [10,20,30,40,50] 3
```

returns: `[10, 20, 30, 50]`

Hint: Consider using Haskell's zip function!

d. (4 points) This problem is independent of parts a - c. Write a Haskell function named *dup_rem* that eliminates consecutive duplicates from a Haskell list (not necessarily in ascending order) and returns a new list that contains only the non-duplicated items. **You must use *map*, *filter* or *foldl/foldr*.** You do NOT need a type signature. Hint: Use Haskell's last fn: last [1,2,4] → 4.

For example:

```
dup_rem [1, 1, 2, 2, 2, 3, 3, 1, 1]
```

returns: `[1, 2, 3, 1]`

4. I'm Partial to Curry(ing) (12 points)

For this problem, consider the following Haskell function:

```
mystery p q [] = False
mystery p q (x:xs) =
  j
 where
  h = length xs     -- Line A
  i = q (p x)
  j = mystery p q xs || i > h
```

a. (5 points) Your first job is to figure out the type signature for the *mystery* function by analyzing its code and performing type inference. Write the **uncurried type signature** (just like Haskell would show it with :t) for this function, using type variables if necessary. You do not need to include type classes in your type signature.

b. (2 points) Now show the **fully curried type signature** for this function, based on the answer you got for part a.

c. (3 points) If we executed the following line of code which uses our mystery function:

```
enigma = mystery (\x -> x `div` 5) (\y -> y^2)
```

what would the **fully curried** type signature be for *enigma*? Write it here:

d. (2 points) Referring back to our original *mystery* function, if you changed Line A to:

```
h = if (elem x "foobar") then 0 else 1
```

you will be able to come up with a more specific type signature. Show the new **uncurried type signature** for the updated *mystery* function here:

5. Eggert's Scoping and Typing (12 points)

Professor Eggert has decided to invent a new programming language, called Egged, and has finalized the syntax and chosen pass-by-object reference for parameter passing like Python. But… he has yet to decide upon Egged's typing system and scoping rules.

He has chosen four potential options for Egged:

- Static typing (w/type inference) and lexical scoping
- Static typing (w/type inference) and dynamic scoping
- Dynamic typing and lexical scoping
- Dynamic typing and dynamic scoping

Prof. Eggert would like to evaluate the behavior of the following Egged program relative to each of the above typing/scoping combinations before he formally picks an option:

```
var x = 42            // Defines a global variable

fn foo(x):
   print(x)
   x = "egg"          //  = sets the value of the variable in scope

fn bletch():
   print(x)           // prints output and then a newline
   x = "emacs"

fn bar():
   var x = "ocaml"     // Defines a local variable
   foo(x)
   bletch()
   print(x)

fn main():
   bar()
   print(x)
```

Let's help Professor Eggert figure out what output his program will produce assuming we adopt each of the following typing/scoping approaches:

a. (3 points) **Static typing** *and* **lexical scoping:** what will the above program's output be, or will it result in a compile/runtime error? If it results in an error, why?

b. (3 points) **Static typing** *and* **dynamic scoping:** what will the above program's output be, or will it result in a compile/runtime error? If it results in an error, why?

c. (3 points) **Dynamic typing** *and* **lexical scoping:** what will the above program's output be, or will it result in a compile/runtime error? If it results in an error, why?

d. (3 points) **Dynamic typing** *and* **dynamic scoping:** what will the above program's output be, or will it result in a compile/runtime error? If it results in an error, why?

6. This 'n' That (12 points)

a. (3 points) For this problem, you're going to write a list comprehension for use within a Haskell function named *everyOther*. The *everyOther* function takes in an input string and returns a new string composed of every other character from the original string, starting with the first character.

For example:

```
everyOther "Hello World!"
```

returns:

```
"HloWrd"
```

Hints: A zip-py solution is the simplest solution. You may find Haskell's *even* or *odd* functions useful.

Write just the Haskell list comprehension that can be used in *everyOther* in the brackets below:

```
everyOther s =
    [                                                                    ]
```

b. (2 points) Assuming we execute the Python *main()* function below:

```
def bar(m):
    return lambda x: m*x

def main():
    m = 2
    f = bar(m)
    m = 5
    print("The answer is: ", f(10))
```

What will this program print?

c. (5 points) For this problem, we will list a series of operations. Your job is to determine whether each operation could reasonably be used in a language that is (a) statically typed, (b) dynamically typed, (c) gradually typed, or some combination of a, b, and c. For each operation, **circle all language typing systems that are compatible with the operation**:

i. Typecasting a Person object to a Dog

Static Typing Dynamic Typing Gradual Typing

ii. Coercing a value like 5 during assignment, as in a = 5

Static Typing Dynamic Typing Gradual Typing

iii. Coercing a variable x in an expression as in x * 5.0

Static Typing Dynamic Typing Gradual Typing

iv. Checking an operation for type-safety at runtime (e.g., x.quack())

Static Typing Dynamic Typing Gradual Typing

v. Performing duck typing

Static Typing Dynamic Typing Gradual Typing

d. (2 points) Consider the following C++ program:

```
int main() {
   int *arr = new int[100];
   delete [ ] arr;
}
```

Describe in one sentence how the *arr variable's* lifetime and scope are affected by the delete command: