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)

Point of Contact: Bonnie/Avii

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:

Ans v1:

a. Ans: ["Bonnie", "Carey"]
b. Ans: ["Bonnie", "Carey", "Brian"]
    i. If student put ["Bonnie"] for part a, give full credit for ["Bonnie", "Brian"] for this question.
c. Ans: ["Bonnie", "Carey"]
d. Ans: ["Bonnie", "Carey", "Brian", "Andrey"]
    i. If student put ["Bonnie"] for part a, give full credit for ["Bonnie", "Brian", "Andrey"]
e. Ans: ["Bonnie", "Carey"]

Ans v2:

a. Ans: ["Liu", "Carey"]
b. Ans: ["Liu", "Carey", "Brian"]
    i. If student put ["Liu"] for part a, give full credit for ["Liu", "Brian"] for this question.
c. Ans: ["Liu", "Carey"]
d. Ans: ["Liu", "Carey", "Brian", "Cui"]
    i. If student put ["Liu"] for part a, give full credit for ["Liu", "Brian", "Cui"]
e. Ans: ["Liu", "Carey"]

Grading Rubric:

- 2 points for every correct answer
- No partial credit beyond what's stated above for error-carry forward
- If a student on exam v1 has the words Liu or Cui, please report an incident of cheating to Carey

- If a student on exam v2 has the words Bonnie or Andrey, please report an incident of cheating to Carey

2. Algebraic Data Type Silliness (15 points)

Point of Contact:s Carey/Brandon

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

Ans v1: (Node 1 (Node 2 (Node 2 (Node 3 (Node 3 Nil)))))
or Node 1 (Node 2 (Node 2 (Node 3 (Node 3 Nil))))

Ans v2: (Node 1 (Node 2 (Node 2 (Node 44 (Node 44 Nil)))))
or Node 1 (Node 2 (Node 2 (Node 44 (Node 44 Nil))))

Grading Rubric:

- 2 points for a perfectly correct answer
- 1 points for just one mistake but otherwise correct
- Use List instead of Node, 0 points
- If they forget a node, give 1 point
- If they get Nil wrong, give 1 point
- If a student on exam v1 has a value of 44 in this answer, please report cheating to Carey
- If a student on exam v2 has a value of 3 in this answer, please report cheating to Carey

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

Ans for v1 and v2:

```
dup_rem :: List -> List
dup_rem Nil = Nil
dup_rem (Node v next) =
  snd (remh (Node v next))
 where
  remh Nil = (0, Nil)
  remh (Node v rest) =
    if v == nextv then (v, n) else (v, (Node v n))
   where
    (nextv, n) = remh rest
```

Alternatively they could use guards for the remh helper function, replacing it with:

```
  remh Nil = (0, Nil)
  remh (Node v rest)
   | v == nextv = (v, n)
   | otherwise = (v, Node v n)
   where
    (nextv, n) = remh rest
```

Here's another variation

```
dup_rem Nil = Nil
dup_rem (Node v Nil) = Node v Nil
dup_rem (Node v1 (Node v2 next))
  | v1 /= v2 = Node v1 (dup_rem (Node v2 next))
  | otherwise = dup_rem (Node v2 next)
```

Another variation

```
ll_to_l :: List -> [Int]
ll_to_l Nil = []
ll_to_l (Node x xs) = [x] ++ (ll_to_l xs)

l_to_ll [] = Nil
l_to_ll (x:xs) = Node x (l_to_ll xs)

check_dup accum val
    | accum == [] = accum ++ [val]
    | val /= (last accum) = accum ++ [val]
    | otherwise = accum

dup_rem :: List -> List
dup_rem link_lst = l_to_ll (foldl check_dup [] (ll_to_l link_lst))
```

Another variation

```
dup_rem :: List -> List
dup_rem lst = reverse_list (dup_rem_helper lst Nil) Nil

dup_rem_helper Nil output = output
dup_rem_helper (Node val nxt) Nil = dup_rem_helper nxt (Node val Nil)
dup_rem_helper (Node val nxt) (Node val2 nxt2) =
    if val == val2 then dup_rem_helper nxt (Node val2 nxt2)
    else dup_rem_helper nxt (Node val (Node val2 nxt2))

reverse_list Nil acc = acc
reverse_list (Node val nxt) acc = reverse_list nxt (Node val acc)
```

Another variation

```
dup_rem :: List -> List
dup_rem Nil = Nil
```

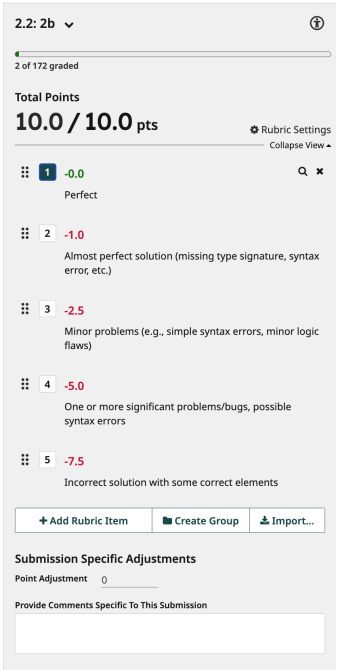
```
dup_rem (Node val Nil) = Node val Nil
dup_rem lst = convert_back (filter_pairs (zip (convert_to_ints lst) (tail
(convert_to_ints lst) ++ [0])))
    where
        filter_pairs :: [(Int, Int)] -> [Int]
        filter_pairs [] = []
        filter_pairs [x] = [fst x]
        filter_pairs ((a, b):xs)
            | a /= b    = a : filter_pairs xs
            | otherwise = filter_pairs xs

convert_to_ints :: List -> [Int]
convert_to_ints Nil = []
convert_to_ints (Node val next) = val : (convert_to_ints next)

convert_back :: [Int] -> List
convert_back [] = Nil
convert_back (x:xs) = Node x (convert_back xs)
```

Grading Rubric:

Proposal - use this and make sure to add comments with specific issues that were wrong



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.**

Ans: 6 new nodes will be created. Why? because we have to reconstruct the whole list as we return from recursion since nodes/ADTs are immutable, leaving out one of the 10 values

Grading Rubric:

- 2 points for 6 new nodes
- 1 points for 5 or 7 new nodes
- 1 point for explaining how a new node is created for every node in the new linked list.
- 0 points for all other answers

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

Point of Contact: Bonnie/Avii/Brandon

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:

Ans:

```
extract2nd :: [(a, b)] -> [b]
extract2nd lst = map (\(_, y) -> y) lst
```

or

```
extract2nd :: [(a, b)] -> [b]
extract2nd lst = map (\(x, y) -> y) lst
```

Grading Rubric:

- 1 point for a valid type signature

- 1 point for proper implementation
- .5 point off for a minor error
- Any variable names are OK

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")]`

Ans:

```
filterBy1st lst i = filter (\(x, _) -> x /= i) lst
```

or:

```
filterBy1st lst i = filter (\(x, y) -> x /= i) lst
```

or:

```
lst i = filter (\(x, y) -> not (x == i)) lst
```

Grading Rubric:
- 2 points for proper implementation; it's Ok if the student writes != or \= instead of /=
- .5 point off for a minor error
- Any variable names are OK

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!

Ans:

removeElemAtIndex lst i = extract2nd (filterBy1st (zip [0..((length lst) - 1)] lst) i)

Grading Rubric:

- 4 points for proper implementation
- .5 point off for each minor error
- Any variable names are OK

d. (4 points) This one might sound familiar :). 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 to include the type signature for full credit.

For example:

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

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

Ans v1:

```
dup_rem = foldl skipDuplicates []
  where
    skipDuplicates [] x = [x]
    skipDuplicates acc x
      | last acc == x = acc
      | otherwise     = acc ++ [x]

dup_rem = foldl skipDuplicates []
  where
    skipDuplicates [] x = [x]
    skipDuplicates acc x = if last acc == x then acc else acc ++ [x]


or

dup_rem = foldr skipDuplicates []
```

```
  where
    skipDuplicates x [] = [x]
    skipDuplicates x acc
      | x == head acc = acc
      | otherwise     = x : acc

dup_rem = foldr skipDuplicates []
  where
    skipDuplicates x [] = [x]
    skipDuplicates x acc = if x == head acc then acc else x : acc
```

Grading Rubric:

- Note: students may use if statements instead of guards
- 1 point for a proper top-level function def, e.g.:
  - remdup = foldl reducer_func []
- 1 point for a valid reducer function that handles the empty list case:
  - for foldr: skipDuplicates x [] = [x]
  - for foldl: skipDuplicates [] x = [x]
- 2 points for a valid reducer function that handles the non empty list case, e.g.:

```
    skipDuplicates x acc
      | x == head acc = acc
      | otherwise     = x : acc
```

- .5 points off for each small error, e.g.:
  - mixing up order of x and acc to foldl/foldr
- The func name may be remdup or dup_rem

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

Point of Contact: Andrey/Gan

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 Hasell 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.

Ans v1:
mystery :: (a -> b) -> (b -> Int) -> [a] -> Bool

Ans v2:
magic :: (a -> b) -> (b -> Int) -> [a] -> Bool

Grading Rubric:

- 5 points for a fully correct answer
- 1.5 points off for each incorrect sub-part, e.g., (b -> Bool) for the second parameter
- 3 points off if their answer is correct but curried in some way
- If you see the word magic on exam v1 please report cheating to Carey
- If you see the word mystery on exam v2 please report cheating to Carey

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

Ans v1 (make sure to do error carry forward for part a):
mystery :: (a -> b) -> ((b -> Int) -> ([a] -> Bool))

Ans v2:
magic :: (a -> b) -> ((b -> Int) -> ([a] -> Bool))

Grading Rubric:

- 2 points for a fully-correct answer
- 1 point off for a single minor mistake
- If you see the word magic on exam v1 please report cheating to Carey
- If you see the word mystery on exam v2 please report cheating to Carey

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:

Ans for v1 and v2 (same):

[Int] -> Bool
or
[Integer] -> Bool
or
[Num] -> Bool

Grading Rubric:

- 3 points for a fully-correct answer
- 1.5 point off for each minor mistake

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:

Ans v1:
mystery :: (Char -> b) -> (b -> Int) -> [Char] -> Bool
or
mystery :: (Char -> b) -> (b -> Int) -> String -> Bool

Ans v2:
magic :: (Char -> b) -> (b -> Int) -> [Char] -> Bool
or
magic :: (Char -> b) -> (b -> Int) -> String -> Bool

Grading Rubric:

- 2 points for a fully-correct answer
- 1 point off for each minor mistake
- If you see the word magic on exam v1 please report cheating to Carey
- If you see the word mystery on exam v2 please report cheating to Carey

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

Point of Contact: Justin/Hanchen

Professor Eggert has decided to invent a new programming language, called Egged, and has finalized the syntax. 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:

Grading Rubric:

- 1.0 points off for each sub-answer which is incorrect (e.g., for b, an answer of ocaml, ocaml, 42, 42 would be 2 points.)
- If you see vim or the number 99 in exam v1 then please report cheating to Carey
- If you see ocaml or the number 42 in exam v2 then please report cheating to Carey

v1 Answers:

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?

Ans: does not compile because we're reassigning x to a string type when x is an integer.

If they indicate that the language might do an implicit cast from string to int, then this could run. Please contact your TA for details

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?

Ans: ocaml, ocaml, emacs, 42

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?

Ans: ocaml, 42, ocaml, emacs

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?

Ans: ocaml, ocaml, emacs, 42

v2 Answers:

a. (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?

Ans: ocaml, ocaml, vim, 99

b. (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?

Ans: does not compile because we're reassigning x to a string type when x is an integer.

c. (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?

Ans: ocaml, ocaml, vim, 99

d. (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?

Ans: ocaml, 99, ocaml, vim

6. This 'n' That (12 points)

Point of Contact: Brian/Justin

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 =
    [                                                                    ]
```

Ans v1 and v2:

[c | (i, c) <- zip [0..] s, even i]

Grading Rubric:

- 3 points for a correct solution
  - Needs to iterate through all items of s
  - Needs to take every other element of the string (e.g. use func ‘even’ or some modulo 2 logic)
  - May use indexing operator (i.e. !!)
- 2 points off if the looping is unpaired
- 1.5 points off for opposite indexing
- 1.5 points off if they have a somewhat correct solution but did not use list comprehension
- .5 point off for each minor error

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?

Ans v1: 20

Please report cheating to Carey if you see: 40

Ans v2: 56

Please report cheating to Carey if you see: 50

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

Ans v1:

i. Typecasting a Person object to a Dog

Static Typing Gradual Typing

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

Static 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

Dynamic Typing Gradual Typing

Ans v2:

i. Performing duck typing

Dynamic Typing Gradual Typing

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

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

Static Typing Gradual Typing

v. Typecasting a Person object to a Dog

Static Typing Gradual Typing

If a person says none of the above for v you can give them full points for this.

Grading Rubric:

- 1 point for a fully-correct answer for each line
- 0 total points for an answer that chooses the diametrically opposite approach (e.g., static for an answer that should be dynamic, or visa-versa)
- .5 point off if the student includes only one of static or dynamic typing when both are required.
- .5 point off for each line if the student fails to circle Gradual Typing when it is required or when it should not be used

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:

Ans: Neither are affected by the delete command.

Grading Rubric:

- 2 points for a fully-correct answer
- 0 points for all other answers