CS131 Fall '24 Midterm

Nov 7th, 2024

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: Haskell basics (15m)	/15
Problem #2: Algebraic Data Type silliness (15m)	/9
Problem #3: Curry, anyone? (15m)	/12
Problem #4: Object references: I told you so! (20m)	/12
Problem #5: Didja really do the project? (15m)	/14
Problem #6: I'm not your type… or am I? (10m)	/8
Total	**/70**

1. [15 points] In this problem, you are to complete a function named `totalVowels` that takes a list of words and returns the total number of vowels (`'a'`, `'e'`, `'i'`, `'o'`, `'u'`, both uppercase and lowercase) present in all the words combined. For example:

```
let words = ["Hello", "World", "HaskEll"]
totalVowels words -- Should return 5
```

Fill in the blanks in the code below to implement this function:

```
totalVowels :: _____________ -> _______________
totalVowels words = helper ___________ 0
  where
    helper [] _____________ = _____________
    helper (_______________) accum =
      helper _________ (____________ + countVowels ___________)

    countVowels _______________ = 0
    countVowels (x:xs)
      | __________ `elem` "____________________" =
        _____________ + _______________ xs
      | otherwise =  ______________________________________
```

Hints:

1. The *helper* function uses an accumulator to compute its result
2. The *countVowels* function operates on a single string at a time
3. Figure out your answer first below, then write your final answer in the slots above

2. [9 points] Consider the following Algebraic Data Type representing a government record. Right now, our government record ADT has only a single variant which represents a document. Each document has a name, a boolean indicating whether the document is top-secret, and zero or more references to related background records:

```
data GovtRecord =
  Document {
    name :: String,
    secret :: Bool,
    relatedDocs :: [GovtRecord]
  }
```

Fill in the blanks below to complete a function named `docExists` that takes a `String` (a name of a document) and a `GovtRecord` ADT that represents a top-level document. The function must check if the top-level document or any documents it directly or indirectly refers to has a matching name, and return `True` if at least one non-secret document matches, and `False` otherwise.

The function should skip the secret documents themselves (i.e., it will never return true if the searched for name *only* matches the name of a secret document), but it should still search all of each secret document's referred-to documents to see if they match the name and are not secret. If multiple documents share the same name, the function must return `True` if at least one non-secret document matches. Your function must use pattern matching. **There will be no loops or cycles in document references.**

Here's how your function might be used:

```
doc1 = Document "Team Members" False []
doc2 = Document "Top Secret Design Doc" True []
doc3 = Document "Top Secret Project Schedule" True [doc1]
mainDoc = Document "Project Orion" False [doc2, doc3]

docExists "Top Secret Design Doc" mainDoc  -- This should return False
docExists "Team Members" mainDoc           -- This should return True
docExists "Project Orion" mainDoc          -- This should return True
docExists "Project Gemini" mainDoc         -- This should return False
docExists "CS131 Final" doc1               -- This should return False
```

Hints:

1. *or [ ]* evaluates to false
2. You may use any built-in Haskell functions, including higher-order functions like *foldl*, *map*, *filter*, and *reduce* in your solution

Fill in the blanks in the code below to implement this function:

```
docExists :: ___________ -> GovtRecord -> ___________
docExists docName (___________ name ___________ related)
  | name == ___________ && secret == ___________ = True
  | _______________ =
      or (___________ (docExists ___________) related)
```

3. [12 points]

Part A. [2 points] Write a complete, syntactically correct type signature for the following function assuming x and y are Doubles. You must write your type signature such that it has no unnecessary parentheses.

```
bar x y = x > (y+5.0)
```

ANSWER: _____

Part B. [3 points] Determine the type of the *boo* function and write a type signature for it.

```
boo = map (\x -> [x])
```

ANSWER: _____

Part C. [4 points] Write a complete, syntactically correct type signature for the following function. You must write your type signature such that it has no unnecessary parentheses.

```
goo a b c =
  head (map c a) > (b ++ "!")
```

ANSWER: _____

Part D. [3 points] As we learned in class, Haskell automatically curries all functions for us. Rewrite the boo function to explicitly show its curried form. Just show the function's body - you may omit the type signature.

```
boo p q r = foldl r 0 p > q
```

ANSWER:

```
boo p = ____________________________________________________________________
```

4. [12 points] Suppose we run this Python program full of lovely object references:

```
from copy import copy, deepcopy
class MyClass:
 def __init__(self, lst, tup, flag):
   self.data = lst
   self.nested_data = tup
   self.flag = flag

 def modify(self, x):
   x[1][1] = "salad"

lst = [5, [10, "corn"]]
tup = ("foo", [12, "milk"])
flag = True

ins1 = MyClass(lst, tup, flag)
ins2 = ins1
ins3 = copy(ins1)
ins4 = deepcopy(ins1)

# PART A
ins1.flag = False
print(ins2.flag)
print(ins3.flag)
print(ins4.flag)

# PART B
ins1.data = [0, [10, "fries"]]
print(ins2.data[0])
print(ins3.data[0])
print(ins4.data[0])

# PART C
ins1.data[1][1] = "taro"
print(ins2.data[1][1])
print(ins3.data[1][1])
print(ins4.data[1][1])

# PART D
ins1.modify(ins1.nested_data)
print(ins2.nested_data[1][1])
print(ins3.nested_data[1][1])
print(ins4.nested_data[1][1])
```

For each part, write out what the program outputs assuming it runs from top to bottom.

Part A. [3 Points]

ANSWERS:

_____

_____

_____

Part B. [3 Points]

ANSWERS:

_____

_____

_____

Part C. [3 Points]

ANSWERS:

_____

_____

_____

Part D. [3 Points]

ANSWERS:

_____

_____

_____

5. [14 Points]

Alex Fife is about to implement **variable scoping** in his Brewin interpreter. To do this, he is considering using a ***stack*** data structure where each element on the stack is an instance of the ***Block*** class:

```
class Block:
  def __init__(self):
    self.vars = {} # maps var names defined in current block to values
```

Here's how Alex is thinking of using his stack structure:

- A new **Block** object will get pushed to the stack each time a new block is entered in the Brewin program being interpreted. The top (i.e., most nested) block will always be in the last element of self.scope_stack, e.g., in self.scope_stack[-1].

- The top **Block** on the stack will be popped each time the Brewin program exits a block.

- Each time a new variable is defined in Brewin (e.g., var x;), the interpreter will add a mapping from the variable's name to **None** to the **vars** dictionary in the top **Block** on the stack.

- When an in-scope variable is assigned to a value (e.g., x = 5;), the **vars** dictionary entry in the appropriate **Block** will be updated so the variable maps to the specified value.

Below is the initial implementation of Alex's interpreter:

```
class Interpreter:
  def __init__(self):
    self.scope_stack = [] # stack of Block objects

  def run_assign(self, var_name, var_val):
    for block in self.scope_stack[::-1]: # [::-1] yields a reversed list
      if var_name in block.vars:
        block.vars[var_name] = var_val
        return True
    return False

  def run(self, program):
    ... # assume this implements Alex's plan from above
```

Here's a Brewin program that Alex will use to test his interpreter. **There are no typos, we checked!**

```
[01] func main() {
[02]   var q;
[03]   q = true;
[04]   var y;
[05]
[06]   for (y = 0; y < 2; y = y + 1) {
[07]     var y;
[08]     y = false;
[09]     if (q) {
[10]       q = y;
[11]       print(q); /* A.3 */
[12]     }
[13]   }
[14]
[15]   foo();
[16]   print(q == y); /* A.4 */
[17] }
[18]
[19] func foo() {
[20]   q = 2;
[21] }
```

Part A. [8 points]

Suppose you run the Brewin program ***with Alex's interpreter as it is currently implemented*** and pause execution right after **line 8** has finished running **during the first iteration of the loop**.

A.1. What is the value of **self.scope_stack[-1].vars** at this point?

ANSWER: _____

A.2. What is the value of **self.scope_stack[0].vars** at this point?

ANSWER: _____

Suppose you run the Brewin program ***with Alex's interpreter as it is currently implemented***. For each line A.3 and A.4, write what the program will output at that line or write "error" if you think the program crashes before it can output that line's print statement.

A.3.

ANSWER: _____

A.4.

ANSWER: _____

Part B. [4 points]

After running the above Brewin program with his interpreter, Alex has spotted an issue with his scoping implementation. To fix it, he's decided to add an additional argument to his **Block** class constructor as follows:

```
class Block:
  def __init__(self, is_func):
    self.vars = {} # dict mapping the names of vars defined in the current block
    self.is_func = is_func
```

Using this new parameter, he's updated his interpreter implementation in the following way:

- If a newly created **Block** is for a *function*, it will be initialized with is_func=True. Otherwise, it will be initialized with is_func=False (e.g., when an *if* block or *for* block is entered).

The last thing Alex needs to do is update his **run_assign** function to take advantage of this new parameter.

Help Alex update his **run_assign** function such that his scoping implementation works properly by **filling in the blanks**:

```
def run_assign(self, var_name, var_val):
  for block in self.scope_stack[::-1]:

    if _____________________ and _____________________:
      return False

    if var_name in block.vars:
      block.vars[var_name] = var_val
      return True

  return False
```

Part C. [1 point]

Assuming the above implementation is correct, write the line number from the original Brewin program where the behavior of Alex's updated interpreter diverges from his initial version.

ANSWER: _____

Part D. [1 point]

What would be the behavior of the updated interpreter at this line? Answer with a single short sentence or phrase. (Again, assume that this updated interpreter correctly implements Brewin.)

ANSWER: _____

6. [8 Points]

Consider the following program in an unknown language:

set.src

```
class Set {
  ... // defines a set class
  func length() {
    // return int representing num items in set
  }
}
```

main.src

```
import set

func goo(y = true) {    // y defaults to true if no parameter is passed
  print("goo: ", y)
}

func foo(x)  {
   goo(x.length())
}

func main() {
  s := Set()  // Creates an empty set
  foo(s)      // Line A
}
```

Part A: (4 points)

Let's assume this program executes successfully. You may also assume that this mystery language performs type coercions (when required) from int to boolean, with 0 coercing to false, and non-zero values coercing to true.

A.1. [2 points] Assuming this language is **statically typed**, what **would the output** of the program be? If it cannot be statically typed, write "cannot be statically typed".

ANSWER: _____

A.2. [2 points] Assuming this language is **dynamically typed**, what **would the output** of the program be? If it cannot be dynamically typed, write "cannot be dynamically typed".

ANSWER: _____

Part B: (2 points)

Imagine that we made the following **one line addition** to our code, with all other constraints left as-is:

main.src

```
import set

func goo(y = true) {    // y defaults to true if no parameter is passed
  print("goo: ", y)
}

func foo(x)  {
   goo(x.length())
}

func main() {
  s := Set()  // Creates an empty set
  foo(s)      // Line A
  foo("abc")  // LINE THAT WAS ADDED!
}
```

Assuming this updated program executes successfully, given only the concepts we've learned so far in class, could this language be **statically typed** or **dynamically typed** or **either**? List all that apply.

ANSWER: _____

Part C: (2 points)

Let's assume the compiler/interpreter refuses to accept the changed program above and instead generates a type error prior to the program's execution. If we comment out **line A** and the program now compiles/interprets successfully, what would the program print out?

ANSWER: _____