CS131 Midterm #1 Solutions/Rubric

Problem #1: 13 points

This problem is all about your ability to use algebraic data types to implement immutable data structures in Haskell/functional languages.

a. (2 points)

- 2 points for either of the following solutions:

```
delete:: List -> Int -> List
delete:: List -> (Int -> List)
```

b. (9 points)

- 9 points for any correct solution that doesn't use an external helper function (if there is a local function defined with let, that's OK). Here are several options, but any correct solution is fine:

```
delete Nil value = Nil
delete (Node nodeval next) value
 | nodeval == value = delete next value
 | otherwise = (Node nodeval (delete next value))
```

Or:

```
delete Nil value = Nil
delete (Node nodeval next) value =
        if nodeval == value then delete next value
        else (Node nodeval (delete next value))
```

- 7 points for a correct solution that uses an external helper function

- 4 points for a solution that just deletes the *first occurrence* of the specified value:

```
delete Nil value = Nil
delete (Node nodeval next) value
 | nodeval == value = next
 | otherwise = (Node nodeval (delete next value))
```

- 1 point off for every syntax error
- For all other semantic errors, deduct points proportional to how impactful the error is

c. (2 points)

The point here is that EVERY node needs to be recreated given how pattern matching works with linked lists, even if no nodes are actually deleted!

- If the student had the correct solution for part B (which deleted all matching nodes), then the following answer gets full credit:
    - 4 new Nodes are generated (or all of the non-deleted nodes are regenerated)
- If the student said all of the nodes will be regenerated, that is acceptable: 2 points
- If the student treated Nil as a Node (it's a variant of List, but not technically a Node - it's a sister to Node) then:
    - If they explicitly stated that there are 5 new List objects created (which could be Node or Nil) then they get 2 points
    - If they just specified 5 new Nodes were created: 0 points
- If the student had a solution that only deletes the first matching node, then the following answer gets full credit for part c:
    - MT VARIATION #1: 4 new Nodes are generated
    - MT VARIATION #2: 1 new Nodes are generated
- All other answers get a zero

Problem #2: 10 points

This problem tests your knowledge of object references, with an example from Python. Answers are slightly different for each midterm; variation are shown below.

a. (5 points)

   MT VARIATION #1:

   [3, [], 10, 100]
   [3, [], 10]
   [3, [33], 10, 100]
   [3, [33], 21]

   MT VARIATION #2:

   [3, [], 92, 100]

```
[3, [], 92]
[3, [44], 92, 100]
[3, [44], 103]
```

- Take 1.25 points off for each incorrect line.
- Take 0.5 points off for missing brackets on the side of the output and/or commas
    - note missing an empty list, such as [], in the output is not covered by the above statement

b. (5 points)

MT VARIATION #1:

```
[3, [], 10, 100]
[3, [], 10, 100]
[3, [33], 21, 100]
[3, [33], 21, 100]
```

MT VARIATION #2:

```
[3, [], 92, 100]
[3, [], 92, 100]
[3, [44], 103, 100]
[3, [44], 103, 100]
```

- Take 1.25 points off for each incorrect line.
- Take 0.5 points off for missing brackets and/or commas

Problem #3: 4 points

You may already know that C++ is weakly-typed, but let's talk about how we can infer it from this question.

The key insight here is that `printStuff` is performing an implicit, unsafe cast: we're reinterpreting a `void*` pointer, which could point to *literally anything*, to a `char*` pointer (often used as a string). This should ring alarm bells in your head: the space the `void*` pointer points to may not mean anything when interpreted as a `char*`! What `printf` does with this information is undefined behavior.

We get confirmation of this with the call to `generateRandomDouble`; we're taking a pointer to something that is clearly not a character (or a string), trying to interpret it as one, and print it!

The core issue is that C is **weakly-typed**. C's compile-time type checking couldn't catch this error, and undefined behavior means that bad code can *sometimes* work.

a. (3 points) - same for both versions of the midterm

- 2 points for any answer that indicates that the undefined behavior is due to C++'s weak typing system
- 1 point for indicating that the problem is a cast between incompatible types, or that we are reinterpreting bits between different types
    - [some students thought that having a void* pointer turning into a double* pointer would cause an error; did not give them points for this]
- 1 point off if the student includes other answers that are not relevant (e.g., that the language uses static typing, etc.) in addition to the correct solution

b. (1 point) - same for both versions of the midterm

- 1 point for indicating that this is using a typecast and not a conversion; we are just reinterpreting the bits of the original double, but no converted value is created

Problem #4: 7 points

Assuming we had version 1 of the midterm, the code would read:

```
v := 42.1                     // := is a variable declaration
v = 42
fmt.Println(v)                // prints v
fmt.Printf("v's type: %T", v) // prints v's type at this moment
```

1. If the language were statically typed, we would expect "42.0 v's type: float" to be printed.
    - In statically typed languages, variables have a fixed type at assignment, and keep them for their lifetime.
    - When we declare v, we're assigning a float to it; even though we aren't explicitly stating v's type, a statically typed language would infer v's type – also a float.
    - Then, when we reassign it, we need to preserve the type of v. To do this, we need to convert the value to a float; the natural answer here is 42.0.
    - So, when we print v, we print the "converted" value – 42.0.
    - Then, when we print v's type, we would print float.
2. If the language were dynamically typed, we would expect "42 v's type: int" to be printed.

    - In dynamically typed languages, the type associated with a variable is whatever tag the current value has.
    - So, when we declare v, its type is float – just like in part 1.
    - However, when we reassign 42 (an int) to it, the underlying value change also causes v's type change to int.
    - So, when we print v, we just print the associated value – 42.
    - Then, when we print v's type, we would print int.

3. We would expect a type coercion to happen if the language was statically typed.
    - If the language is statically typed, v is always a float. So, when we reassign a value of type int, something needs to happen:
        i. either, the language coerces 42 to something that can fit into a float (what is happening in this example) – ex, 42.0
        ii. or, it errors
    - Contrast this with a dynamically typed language: no coercion, cast, or conversion is necessary. Since variables like v has no fixed type, reassigning a value with a different type to it is totally fine. Think of this like Python!
4. It is using type inference.

Fun fact: this language is Go, and Go is statically typed. It actually does neither of what we outlined above; it has very special behavior with numeric constants. Look it up!

Rubric follows:

a. (2 points)

   MT VARIATION #1

- 1 point for saying it should print "42.0 v's type: float" or any answer indicating that 42 is a floating-point or double number (e.g., 42.000 v's type: double)
- 1 point for saying that the reason is that variable v has a type of floating point/double (including after we perform: v = 42)

   MT VARIATION #2

- 1 point for saying that after the assignment to 99 (v = 99), variable v refers to an integer value and not a floating-point value
- 1 point off if the student says the "type of the v variable" is int, since the variable v has no specific type in a dynamically typed language

b. (2 points)

   MT VARIATION #1

- 1 point for saying it should print ”42 v's type: int”
- 1 point for saying that after the assignment to 42 (v = 42), variable v refers to an integer value and not a floating-point value
- 1 point off if the student says the “type of the v variable” is int, since the variable v has no specific type in a dynamically typed language

  MT VARIATION #2

- 1 point for saying it should print "99.0 v's type: float" or any answer indicating that 99 is a floating-point or double number (e.g., 99.000 v's type: double)
- 1 point for saying that the reason is that variable v has a type of floating point/double (including after we perform: v = 99)

c. (2 points)

  MT VARIATION #1

- 1 point for indicating that a coercion must only happen in a statically typed system
- 1 point for indicating that the value of 42 must be converted from integer to floating point (0 points for calling this a “cast” rather than a conversion)

  MT VARIATION #2

- 1 point for “type inference”

d. (1 point)

  MT VARIATION #1

- 1 point for “type inference”

  MT VARIATION #2

- 1 point for indicating that a coercion must only happen in a statically typed system
- 1 point for indicating that the value of 99 must be converted from integer to floating point (0 points for calling this a “cast” rather than a conversion)

Problem #5: 5 points

Recall our in-class definition of subtype:

A is a subtype of B if and only if:

- Every element in the set of values of A is also in the set of values of B

- An object of type A can be used in any code that requires an object of type B

With this definition, both int and float are subtypes of JuliaNumber.

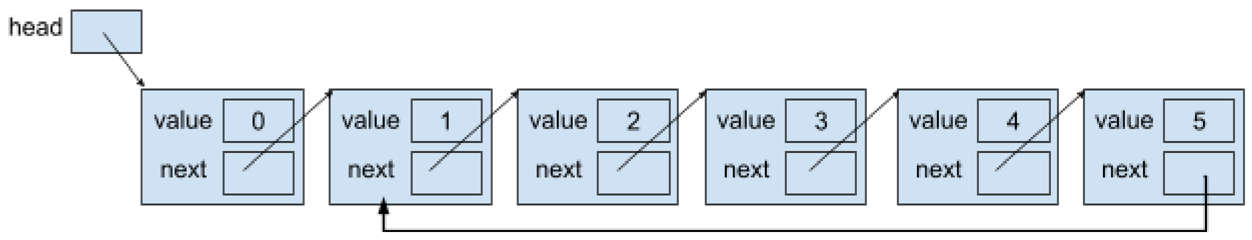
- Every element in the set of values of A is also in the set of values of B
    - All integers can be represented as a JuliaNumber with the decimal portion being .0
    - All floats can be represented as a JuliaNumber with the representation being identical to the float's machine value
- An object of type A can be used in any code that requires an object of type B
    - If there's an implicit conversion from int to JuliaNumber and float to JuliaNumber;
    - And, the above point is true;
    - Then it must be true that anywhere we expect a JuliaNumber, we can pass a int/float and it just works!

(5 points total)

- 2 point for correctly identifying that int is a subtype of JuliaNumber and stating why: all integers can be represented as a JuliaNumber with the decimal portion being .0
- 2 point for correctly identifying that float is a subtype of JuliaNumber and stating why: all floats can be represented as a JuliaNumber with the representation being identical to the float's machine value
- 1 point for providing the full definition of a subtype:
    - Every element in the set of values of A is also in the set of values of B
    - An object of type A can be used in any code that requires an object of type B
- Take 1 point off for every incorrect/extraneous answer provided.

Problem #6: 16 points

This problem creates a circularly linked-list like this:



In a language with mark and sweep or mark and compact garbage collection, once our head object reference is set to nil, there are no "root" references to our linked list (root references being global variables, local variables/parameters, static variables ,etc.), and so the entire list, including the head node, will be garbage collected when there is memory pressure that initiates a garbage collection event. We can't be sure when that GC event will happen, if ever, but if/when there is memory pressure the entire list will be garbage collected.

In a language with reference counting, once our head object reference is set to nil, the head node will have a reference count of zero, so it will be subject to garbage collection. However, the other nodes (1-5) will all have a reference count of 1, since each node refers to the next, and the last node refers to the first. So in a language with strict reference counting (and no fancy enhancements), nodes 1-5 will never be reclaimed. The garbage collection of node 0 might take place immediately once head is set to nil, or might be batched up at a later time if the language batches up its reclamation to avoid cascades of deletions (like we'd have here).

MT VARIATION #1

a. (4 points)

- 4 points for: We'd expect all of the nodes to be reclaimed when a garbage collection event occurs, because there are no active object references to any of the nodes once we run the head = nil statement.
- Take 1 point off for every incorrect/extraneous answer provided.

b. (4 points)

- 2 points for: We can't know when the garbage collection event will take place, or even if it will take place
- 2 points for: Garbage collection will only happen if/when there is memory pressure
- Take 1 point off for every incorrect/extraneous answer provided.

c. (4 points)

- 2 points for identifying the proper node that will be reclaimed: We would expect node 0 to be reclaimed
    - 0 points if they say any other nodes will be garbage collected
- 2 points for the reason: The other nodes are in a cycle which ensures that they all have a reference count of 1, even when the head node is destroyed. Thus the nodes will never reach a reference count of zero.
- Take 1 point off for every incorrect/extraneous answer provided.

d. (4 points)

Either of the following answers is acceptable for 4 points:

- The nodes will immediately be GCed when their reference count reaches zero

- When the nodes' reference counts reach zero, the nodes may be queued up for periodic garbage collection to prevent slow/expensive cascades from running all at once. In this case, the nodes would be collected at some point in the future.
- Take 1 point off for every incorrect/extraneous answer provided.

MT VARIATION #2

a. (4 points)

- 4 points for: We'd expect all of the nodes to be reclaimed when a garbage collection event occurs, because there are no active object references to any of the nodes once we run the head = nil statement.
- Take 1 point off for every incorrect/extraneous answer provided.

b. (4 points)

- 2 points for identifying the proper node that will be reclaimed: We would expect node 0 to be reclaimed
    - 0 points if they say any other nodes will be garbage collected
- 2 points for the reason: The other nodes are in a cycle which ensures that they all have a reference count of 1, even when the head node is destroyed. Thus the nodes will never reach a reference count of zero.

c. (4 points)

Either of the following answers is acceptable for 4 points:

- The nodes will immediately be GCed when their reference count reaches zero
- When the nodes' reference counts reach zero, the nodes may be queued up for periodic garbage collection to prevent slow/expensive cascades from running all at once. In this case, the nodes would be collected at some point in the future.
- Take 1 point off for every incorrect/extraneous answer provided.

d. (4 points)

- 2 points for: We can't know when the garbage collection event will take place, or even if it will take place
- 2 points for: Garbage collection will only happen if/when there is memory pressure
- Take 1 point off for every incorrect/extraneous answer provided.

Problem #7: 10 points

Consider the following Python code, with provided line numbers (version #1 shown here, but both are nearly identical):

```
01   def f(x):
02    a = x
03    def g(x):
04      print(x + a)
05    return g
06
07   def h(x):
08    d = f(3)
09    for i in range(5):
10     d(i)
```

MT Variation #1

a. (3 points)

On line 4, the x variable from line 3 shadows the x variable from line #1, so the g(x) version of x is in scope. Variable a is also in scope. Function f itself is in scope, since otherwise f couldn't call itself, which would be required for recursion. And of course print is in scope. So the answer would be:

functions: f, print
variables: a, x (as in g(x) on line 3)

- Three points for the correct answer
- Take 1 point off for each missing/extraneous answer

b. (4 points)

Local: x	x is a local variable
Enclosing: a	a is a variable from an enclosing scope
Global: f	f is a globally-scoped entity
Built-in: print	print is obviously a built-in function

- 1 point for each correct answer

c. (2 points)

2 points for any of these answers:

- On line 8, when f is called, the parameter x in h is out of scope within f, but is still alive.

- On line 10, when d is called, x, d and i are out of scope, but are still alive

MT Variation #2

a. (3 points)

functions: f, print
variables: que, moo (as in g(moo) on line 3)

- Three points for the correct answer
- Take 1 point off for each missing/extraneous answer

b. (4 points)

Local: moo	moo is a local variable
Enclosing: que	que is a variable from an enclosing scope
Global: f	f is a globally-scoped entity
Built-in: print	print is obviously a built-in function

- 1 point for each correct answer

c. (2 points)

2 points for any of these answers:

- On line 8, when f is called, the parameter goo in h is out of scope within f, but is still alive.
- On line 10, when d is called, goo, doo and eye are out of scope, but are still alive

d. (1 point)

- Shadowing is taking place on line 3 and on line 4
- Either answer (3 or 4) is fine

Problem #8: 15 points

a. Give 2 points for a proper type signature - the type signature is only required for the top-level substr function. Zero points for anything else. Any of these variations is OK:

   ```
   substr :: String -> String -> Bool
   substr :: String -> (String -> Bool)
   ```

```
substr :: [Char] -> [Char] -> Bool
substr :: [Char] -> ([Char] -> Bool)
```

b. Here's one possible solution:

MT Variation #1

```
substr [] [] = True
substr _ [] = False
substr xs (y:ys) = (substr xs ys) || (front_substr xs (y:ys))

front_substr :: String -> String -> Bool    -- Type signature here not required for full points

front_substr [] _ = True
front_substr (x:xs) (y:ys)
  | x == y = front_substr xs ys
  | otherwise = False
```

MT Variation #2

```
substr [] [] = True
substr [] _ = False
substr (x:xs) ys = (substr xs ys) || (front_substr ys (x:xs))

front_substr :: String -> String -> Bool    -- Type signature here not required for full points
front_substr [] _ = True
front_substr (x:xs) (y:ys)
  | x == y = front_substr xs ys
  | otherwise = False
```

- Give 13 points for a fully correct solution
- Take 1 point off for each minor syntax error (e.g., missing parentheses around a pattern match); don't worry about case sensitivity on things like True/False
- Take 3 points off for every base case they're missing
- For all other minor semantic errors, take 2 points off
- For all major semantic errors, take at least 6 points off

Problem #9: 10 points

This problem tests your knowledge of currying and partial application. Recall that in haskell:

```
function:: a -> b -> c
```

is equivalent to:

function:: a -> (b -> c) after currying

All of the variations in the problem may be derived from this rule.

a. (5 points) Answer:

MT Variation #1

    i, iv, vii
    ii, v
    iii, viii
    vi

MT Variation #2

    i, iv, v
    iii, vii
    ii, viii
    vi

- Take 1 point off for each item placed in an incorrect grouping. Only one penalty given for an improper grouping (not two penalties: one for the group where the item is missing, and one for the group where the student incorrectly added it).

b. (2 points)

- 1 point for the first partial application: ((b -> c) -> d) -> e -> f
- 1 point for the second partial application: e -> f

c. (1 point)

If we have a curried function g, then it will take one parameter, and return a function that takes a second parameter. The first parameter must be passed into g, and the second parameter must be passed into the function returned by g.

1 point for: g(x)(y), or anything that is semantically equivalent, e.g.

```
a = g(x)
a(y)
```

d. (2 points)

The point here is to transform a function that takes three parameters into a first function that takes the first parameter, and returns a second function. The second function must take the second parameter, and return a third function. The third function must take the last parameter and return the final result.

MT Variation #1:

```
def curried_f():
        return lambda x: lambda y: lambda z: x * y + z
```

other possible answer:

```
def curried_f():
  def f(x):
     def g(y):
        def h(z):
           return x*y+z
        return h
     return g
  return f
```

MT Variation #2:

```
def curried_f():
        return lambda p: lambda q: lambda r: p * q + r
```

other possible answer:

```
def curried_f():
  def f(p):
     def g(q):
        def h(r):
           return p*q+r
        return h
     return g
  return f
```

- 1 point off for adding a return to the lambda expression, e.g.:

  return lambda x: lambda y: lambda z: **return** x * y + z

- 0 points for this:

  - return lambda x,y,z: x*y + z

- including formal parameters in the signature of curried_f()