CS131 Fall 22 - Midterm #1

Nov 2nd, 2022

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

Student ID: ________________________________

Problem #1	/13
Problem #2	/10
Problem #3	/4
Problem #4	/7
Problem #5	/5
Problem #6	/16
Problem #7	/10
Problem #8	/15
Problem #9	/10
Total	**/90**

Please practice academic integrity - don't cheat!

1. (13 points total)

   Consider the following algebraic data type which defines the variants required for a linked list of integers in Haskell:

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

   For this problem, you will define a function called "`delete`" that accepts two parameters: a linked list and an integer value to delete. The function deletes *all* of the nodes from the linked list that contain the passed-in value.

   Here's how it would be used:

```
some_list = Node 5 (Node 10 (Node 40 (Node 10 Nil)))

# deletes all nodes with a value of 10 from the list
# returns: Node 5 (Node 40 Nil)
delete some_list 10

# deletes all nodes with a value of 42 from the list
# returns: Node 5 (Node 10 (Node 40 (Node 10 Nil)))
delete some_list 42
```

a) (2 points) Provide a valid Haskell type definition for the `delete` function.

b) (9 points) Write the body of the `delete` function in Haskell - you may use any of the syntax/techniques we discussed during class. **You may NOT use any helper functions.**

c) (2 points) Given the following linked list:

```
some_list = Node 17 (Node 5 (Node 10 (Node 40 (Node 12 Nil))))
```

when we call:

```
delete some_list 40
```

How many *new nodes*, if any, will be generated by your delete function?

2. (10 points total)

   Consider the following Python program (with included line numbers for clarity) which performs some operations on lists:

```
01    l1 = [55]
02    l2 = [3, l1, 10]
03    l3 = [3, l1, 10]
04    l2.append(100)
05    l2[1].remove(55)
06    print(l2)
07    print(l3)
08    l3[1].append(33)
09    l3[2] += 11
10    print(l2)
11    print(l3)
```

a) (5 points) What is the output of the above program?

b) (5 points) Suppose we changed line #3 above (**bolded** for clarity) to the following:

```
l3 = l2
```

   What would the output be now?

3. (4 points total)

   Consider the following C++ program:

```
printStuff(void* buff) {
  char* buf = buff;
  cout << buf;
}

double* generateRandomDouble() { /* Generates a random double */ }

int main() {
  printStuff(generateRandomDouble());
}
```

   If you weren't aware, a `void*` pointer can be reinterpreted to a pointer of any type. In C++, this code compiles with no errors.

   If you run this program repeatedly, however, you will find that sometimes it works, sometimes it crashes, and sometimes it keeps on printing nonsense and segfaults!

a) (3 points) Explain, with explicit reference to one of the type systems we learned in class, why this occurs (even though the program compiles).

b) (1 point) Does `printStuff`'s implementation use a cast, a conversion, or neither?

4. (7 points total)

   Classify that language got *such high ratings* that we're even bringing it to your midterm!

   Take a look at code from this mystery language. Assume that it has *different* types for integers (`42`) and floats (`42.1`).

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

   This code works with no errors.

a) (2 points) Assume that the language is **statically typed**. What do you think would be printed, and why?

b) (2 points) Assume that the language is **dynamically typed**. What do you think would be printed, and why?

c) (2 points) In which type system – static, dynamic, neither, or both – would you expect that a type coercion is occurring and why?

d) (1 point) Assuming the language above is statically typed, what mechanism is it using to determine the type of variable v when it is defined?

5. (5 points)

   William is a student in CS 131, and feels like he has a pretty good understanding of why an `int` and `float` are not subtypes of each other.

   William is learning the Julia programming language, which supports *arbitrary-precision* arithmetic. In other words, it has a data type that can represent any decimal number—no matter how large, or how many digits are after the decimal.

   Let's call this data type `JuliaNumber`. Additionally, assume that Julia has some way to (implicitly) convert between `JuliaNumber` and `int`, `float`.

   William's question is this: is either `int` or `float` a subtype of `JuliaNumber` (or both/neither)? Why or why not? **Please explicitly reference the definition of subtype we have discussed in class.**

   For this problem, assume `int` and `float` are fixed-bit representations, like in C++.

6. (16 points total)

   You have been given the following program in a language you're not familiar with:

```
fn create_list(n):
  # { } creates a new object has no fields/properties or methods
  head = cur = { }
  for i in range(n):    # goes from [0,n)
    # obj["field"] = val is the equivalent of obj.field = val
    cur["value"] = i
    next = { }
    cur["next"] = next
    cur = next
  cur["value"] = n
  cur["next"] = head["next"]
  return head

fn main():
  head = create_list(5)
  head = nil
```

   For the following questions, if you need to reference a specific node you may identify it based on the number stored within it (e.g., "node 3").

a) (4 points) Assuming this language is using Mark and Compact garbage collection, which nodes would you expect to be garbage collected sometime after we reach the line "`head = nil`" in main()? Why?

b) (4 points) At what point would the nodes you listed in part a) be garbage collected (using Mark and Compact)? Immediately? At some point in the future? What would determine when this happens?

c) (4 points) Assuming this language is using Reference Counting garbage collection, which nodes would you expect to be garbage collected sometime after we reach the line "`head = nil`" in main()? Why?

d) (4 points) At what point would the nodes you listed in part c) be garbage collected (using Reference Counting)? Immediately? At some point in the future? What would determine when this happens?

7. (10 points total)

Consider the following Python code (with provided line numbers for clarity):

```
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)
```

a) (3 points) Among the following variables and functions:

   Variables: `x` (as in `f(x)` on line 1), `a`, `x` (as in `g(x)` on line 3), `d`, `i`
   Functions: `f`, `print`

   Which are in scope on line 4: (`print(x + a)`)?

b) (4 points) Classify the in-scope functions and variables you listed in part a) according to the LEGB (Local, Enclosing, Global, Built-in) rule in Python (e.g. "`x` is a global variable").

c) (2 points) During the execution of function `h` (including calls to other functions), find and describe an instance where a variable is out of scope but is still alive.

d) (1 point) Is there any shadowing taking place in this code? If so, on what line(s) is shadowing taking place?

8. (15 points total)

   Write a Haskell function named `substr` that determines whether a first string is a substring of a second string. It might be called as follows:

   `substr "ce" "i like ice cream"` should return `True`
   `substr "lice" "i like ice cream"` should return `False`

   You must include a proper Haskell type signature for full credit. You may use helper function(s).

   Hint: Recall that in Haskell, strings are just lists of characters.

9. (10 points total)

a) (5 points) Which of the Haskell function signature types below are equivalent with each other? So, for example, your answer should look like “i and viii are equivalent” and “ii, vi, and vii are equivalent.” Group the similar types together.

i. `a -> b -> c -> d`
ii. `(a -> b) -> c -> d`
iii. `a -> (b -> c) -> d`
iv. `a -> b -> (c -> d)`
v. `(a -> b) -> (c -> d)`
vi. `((a -> b) -> c) -> d`
vii. `a -> (b -> (c -> d))`
viii. `a -> ((b -> c) -> d)`

b) (2 points) You are given the following function with the type signature:

```
function :: a -> ((b -> c) -> d) -> e -> f
```

What is the type signature of the resulting function after 1 partial application? What about after 2 partial applications?

c) (1 point) Suppose we have a Python function named `curry_2` that takes in any Python function expecting exactly 2 arguments and converts it into its equivalent curried form.

   Let `g = curry_2(f)`. Given two variables `x` and `y`, show how you would call `g` on `x` and `y` using valid Python syntax to yield the same value returned by `f(x, y)`.

d) (2 points) Given the Python function:

```
def f(x,y,z):
   return x * y + z
```

   Write a function with the signature `curried_f()` in Python that returns a curried version of the function `f` shown above.