CS 430 Types

Parts 1-3 (only those problems that can’t be answered by PythonTutor)

2. What is the name of the "*" operator in C (e.g., on the two lines after location A), and what does it do?

The dereference operator accesses memory through a pointer variable.

9. Does the program leak memory? Are there any dangling pointers?

It does not leak memory, but there are two dangling pointers (‘a’ and ‘p’) after the call to “free”.

10. Describe how the *locks & keys* approach described in CPL section 6.11.8.2 would help deal with a problem identified in #9.

It would ensure that the access through ‘p’ is flagged as invalid because the keys would no longer match.

11. Describe how *mark-sweep garbage collection* would help deal with a problem identified in #9.

It would remove the need to explicitly free memory, keeping it accessible for the dereference.

12. Describe how *reference counters* would help deal with a problem identified in #9. How many references are there to each 4-byte memory cell allocated on the heap at location C?

It would highlight the fact that there are two pointers to cells in the array, flagging this as a potential problem when the memory is freed using one of the pointers. There’s one pointer to the first cell and one pointer to the second cell.

13. What is a *reference type* and how is it different from a *pointer type*? Does C have reference types, pointer types, or both? What about C++, Java, and Ruby?

A pointer refers to a specific address in memory (regardless of whether it’s a valid object) while a reference refers to a specific object in memory.

C: pointers only	**C++: pointers and references**
Java: references only	**Ruby: references only**

15. What is the minimum number of bytes necessary to store the struct variable ("v") in the above program?

4 (int) + 1 (char) + 8 (double) = 13 bytes total

16. Suppose that on a particular system there are X different possible integers, Y different possible characters, and Z different possible double-precision numbers. As a function of X, Y, and Z, what is the total number of different possible values for the struct variable ("v")?

X * Y * Z

17 (partial). Change "`struct`" to "`union`" in the first line. How does this change what is stored in memory? What is the minimum number of bytes required to store the union variable? As a function of X, Y, and Z (as defined in #15), what is the total number of different possible values for the union variable ("v")?

Only a single value is stored. The minimum storage is now 8 bytes (the maximum of 4, 1, and 8). The total number of possible values is X + Y + Z.

Part 4: Type Equivalence
Suppose the following declarations have been made in a C-like language. For each context, circle all of the assignments that are valid, and **cross out all that are not valid**.

```
typedef float inches;
typedef float feet;
typedef struct { inches x; feet y; } box;
typedef struct { inches x; feet y; } bin;

inches a, b;
feet c;
float d;
struct { inches x; feet y; } m, n;
box o;
bin p;
```

18. Assume assignments require name equivalence unless at least one type is anonymous, in which case they only require structure equivalence.

[✓ a = b]	~~a = c~~	~~a = d~~	~~a = m~~
[✓ m = n]	[✓ m = o]	~~p = o~~	~~m.x = d~~
[✓ m.x = a]	[✓ o.x = a]	~~m.x = m.y~~	[✓ m.x = o.x]

19. Assume assignments only require structure equivalence, regardless of type aliases.

[✓ a = b]	[✓ a = c]	[✓ a = d]	~~a = m~~
[✓ m = n]	[✓ m = o]	[✓ p = o]	[✓ m.x = d]
[✓ m.x = a]	[✓ o.x = a]	[✓ m.x = m.y]	[✓ m.x = o.x]

20. Assume assignments require name equivalence for primitive types and their aliases but permit structure equivalence for non-primitive types.

[✓ a = b]	~~a = c~~	~~a = d~~	~~a = m~~
[✓ m = n]	[✓ m = o]	[✓ p = o]	~~m.x = d~~
[✓ m.x = a]	[✓ o.x = a]	~~m.x = m.y~~	[✓ m.x = o.x]