**Fall 2020 CS 3723 Programming Languages Assignment #1**

Name \_Andrew DiCarlo\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section \_001\_\_\_\_\_\_\_\_ abc123 \_\_ebq929\_\_\_\_

Given the C code shown below, what is the output? You \*must\* run this on the fox machines.

Aristotle is frequently misquoted as saying *"The whole is greater than the sum of the parts.*" A better translation is "*In the case of all things which have several parts and in which the totality is not, as it were, a mere heap, but the whole is something besides the parts, there is a cause; for even in bodies contact is the cause of unity in some cases, and in others viscosity or some other such quality.*"

Size(char) = \_\_\_1\_\_\_

Size(int) = \_\_\_4\_\_\_

Size(void \*) = \_\_8\_\_\_

Size(s1\_t) = \_\_\_3\_\_\_

Why? \_s1\_t is made up of 3 chars, which are each of size 1. Since they are all a single byte, there is no need for padding to make sure they are aligned with a word boundary. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Size(s2\_t) = \_\_\_16\_\_

Why? s2\_t consists of a void \* followed by two chars. A void \* is 8 bytes and a char consists of 1 byte each, for a total of 10 bytes of data. However, the end of the structure does not end up on a word boundary without padding. This means that if a second s2\_t structure is allocated contiguously after the first, the void \* would end up spanning two words. By adding 6 bytes of padding into the structure, this misalignment is avoided

Size(s3\_t) = \_\_\_24\_\_

Why? The first char takes up 1 byte of memory, but to put the void \* on a word boundary 7 bytes of padding is required. This accounts for the first 16 bytes (1 + 7 + 8). The next char element only takes up one byte, but without padding the structure ends misaligned with the word boundary, as with s2\_t. Adding 7 bytes fixes this, for a total size of 24 (1 +7 + 8 + 1 + 7 = 24).

Size(s4\_t) = \_\_\_12\_\_

Why? The first char take up 1 byte of memory, but without padding the int does not land on an alignment boundary. Since an int is only 4 bytes, rather than 8, its alignment requirement is 4 bytes rather than 8. Similar to s2\_t and s3\_t, the structure needs to be padded to prevent misalignment of a following structure. Adding 3 bytes to the char, followed by the 4 byte int, along with the last char and 3 bytes of padding amounts to 12 bytes total (1 + 3 + 4 + 1 + 3 = 12).

