CSCI 312 - Types HW **Nathan C. Owen**

(1) Fundamentally, a language is statically typed if the type of a variable is known during the compiling of the program. This means that the type of this variable may not be altered after its instantiation. A dynamically typed language need not define its variables until runtime, and so the type of the variable may change during the course of the program.

The following sample of code has no type errors when dynamically typed, but does have type errors when statically typed. This is because no types are directly specified.

```
variableOne = 2
variableTwo = 1 * variableOne
```

(2)
(a) 00111101110011001100110011001101
(b) 00111111100000000000000000000000

(3)

```
float *a; \\ This instantiates a pointer, which references a memory address of either
          \\ 4 or 8 bytes, depending on the system environment (32 bit vs. 64 bit)
          \\ a = 32 bits (or potentially 64)

float *b[100]; \\ This instantiates an array of pointers that is of length 100.
               \\ b = 32 bits * 100 pointers = 32,000 bits = 4,000 bytes

union {int x; float y; double z;} c; \\ This instantiates a Union, which is a data structure available
                                     \\ that allows for multiple pieces of data of different types
                                     \\ in the same space in memory - though only one can be present
                                     \\ at a time. Its size accommodates the largest member.
                                     \\ c = 8 bytes (the size of the largest member, the double)

enum day{Monday, Tuesday, Wednesday} d; \\ This instantiates an enumeration, a way to assign names to
                                        \\ integral values. Enumerations should always be the same
                                        \\ size as an int, and so in our this case, d = 4 bytes
```

(4) The difference between the machines is in the way that they read memory. Suppose that a big-endian machine read in the bytes 12 34 56 78. If a little-endian machine where to read in bytes of equivalent value, they would instead follow this arrangement 78 56 34 12. The Intel x86 processor is a common little-endian architecture, and the IBM z/Architecture mainframes are big-endian processors.

(5) The address would be 1584.