Bryan Duong, 013104973
Phillips Huynh, 012904708

CECS 424 Assignment 10 Discussion Report

With Java, it does not have the memory management capabilities that C++ has. To compensate for this, Java has a built-in method that allows the developer to convert a floating point number to its binary representation according to the IEEE754 format. Conversely, Java has a built in method that allows developers to convert binary back to its floating point number. Since Java does not have the “unsigned” keyword, we would use the long data type instead to represent an unsigned integer. If we were to use int, we would lose the precision of the values that we need to display and sometimes encounter numbers that are too “large”. The bitwise operation used in Assn9.cpp can remain the same in our Java translation. Java also has a built-in function “math.ulp(float)” which calculates the distance between a given float value and the next larger float value with the least precision. This allows us to avoid converting a float to its binary representation, incrementing it, and then converting it back when performing the “Floating Point Number Enumerator” section of the assignment. Skipping this step can allow for faster performance. Knowing this, we expect Java should have similar performance with the original file.

With Python, we use the statement np.float32 to represent floating point numbers with IEEE754 32 bit format. Since we cannot manually manage memory in Python, we import struct and use struct.pack and struct.unpack to convert our floating point number to its binary representation. The struct module allows us to perform conversions with Python values and C structs. We pack the number in with float format, in our case ‘>f’, and then unpack the number in long, which is represented by ‘>l’. The ‘>’ character allows us to specify that the byte order

should be big endian, which means that the big end is stored first (first byte represents the primary value). However, since packing and unpacking returns the value as a tuple, we expect the conversion to be slower in Python compared to C++. The bitwise operation used in Assn9.cpp can remain the same in our Python translation. Python variables do not need explicit declarations to reserve memory space so a variable can be assigned any data type values.

An advantage that Javascript has is that all Number types in Javascript are considered as 64 bit floating point numbers. The user would not have to worry about the types of the variables they are using. However Javascript is a high-level language, which means that in order to convert a float to its binary representation we create a buffer to use between a float32 number and an unsigned integer number. Once the buffer is created, we place our number inside the buffer by using a Float32 object and return the buffer with an Uint32 object to get the binary representation of the float. We can reverse this process to get the numeric value of the float back. Although Javascript does use 64 bit floating point numbers, we can use the same bitwise operations that were used in Assn9.cpp to produce the same results. This is because when a bitwise operation is performed, Javascript converts the number to 32 bit signed numbers. After the operation is complete, the result is converted back to Javascript numbers. However this can result in slower performance when performing bitwise operations to get the sign, exponent, and mantissa of a floating point number. Overall we would imagine our Javascript translation of the assignment would be slower compared to the original C++ file.