Holt Crews, jhc6we, 10/31/17, inlab8.pdf

When an int, char, or float is passed by value in C++, the assembly code first pushes rbp to the stack. Then, the rsp register is moved to rbp. Then, the assembly code takes the parameter value and puts it into a spot at [rbp - <type size>]. All of my functions just returned the parameter that was passed in, so that value was then put into eax and then returned. When a pointer is passed into a function, the code takes the pointer and places into a spot that is always [rbp(the same as rsp) - 8]. This [rbp-8] is then also moved into rdi. When the object (a stack from the STL class) was passed in, the stack pointer was first moved to the rbp register and then rdi, the parameter register, was passed to the position [rbp-8]. The rbp register was then popped and the program returned nothing. This makes me think that really a pointer to that object is passed into the function instead of an actual copy of that object as it is very similar to the pointer assembly code.

For an array passed by value into a method, rdi, the parameter register, a pointer that points to the start of the array, is passed into the spot [rbp-8] where rbp is the same as the stack pointer. Then, to access a member of the array, assembly takes rdi and adds the required number to get to that member (e.g. [rdi+4] will get the second element in an array of ints). This array is treated just like a pointer was treated in assembly which aligns with the way in which C++ deals with arrays in general.

When passing by reference in assembly, the code takes the memory address of the value and places into a spot that is always [rbp(the same as rsp) - 8]. This makes sense because memory addresses are all the same size and are not dependent on the data type at that memory address. This [rbp-8] is then also moved into rdi. This is done in the exact same way as when a pointer is passed by value. I would guess the only difference is what assembly code is allowed to be generated and what is automatically generated when using a reference versus a pointer. A reference and a pointer are basically the same thing with a few key differences, and I would guess that these differences are only represented in the assembly code when necessary.