Here is an example of a buffer overflow problem. The buffer here is an array named “buffer.” Notice in Figure 1, the buffer size is 3. In lines 11 – 13, I assign three values to the elements in the array. Then, in line 15, notice the circled 10 – this is reading past the buffer’s bound of 3. Hence an overflow. The execution is in the screen capture to the right of the code in Figure 1, notice, that the first three elements (starting from 0) are 1, 2, and 3, respectively, as assigned in line 11 – 13. Remember a size of 3 means the elements range from 0 the 2 as arrays are zero-based, or start from 0. Notice the other data starting from Element #3 through 19 are values in memory belonging to another process – not this one. That is the problem – the process associated with the running of this program does not “own” that chunk of memory.

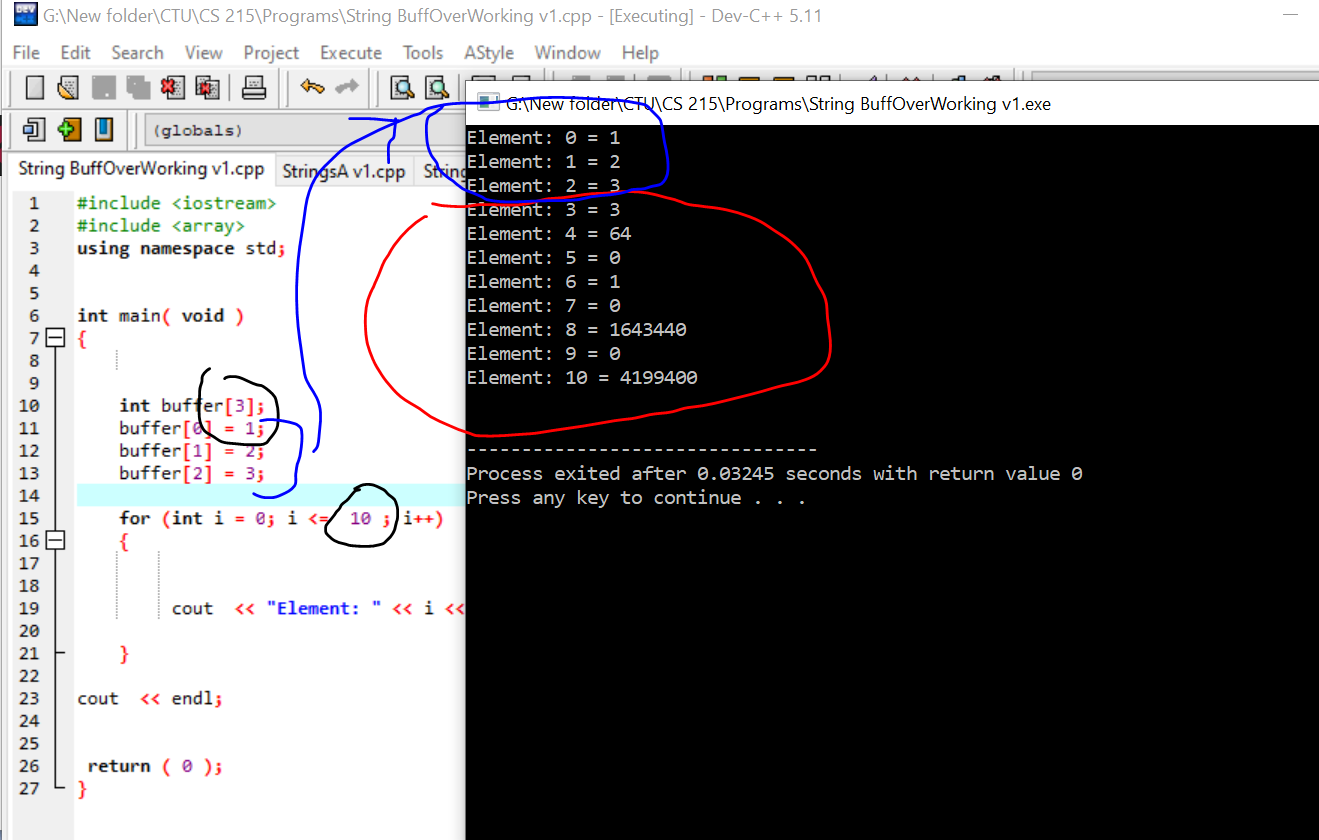


Figure 1: Buffer Overflow: Reading past the buffer

Notice in Figure 2, that we are writing to memory past the range of the buffer. In line 23, we are going to an element number of 5 but we defined the array (buffer) at 3 values. In the execution on the right-hand side, in Figure 2, you can see that we have put data into those places that we don’t own. The memory addresses of each are provided. Notice the addresses of the buffer’s elements are the same – as it is an array. In line 10, 12, and 27 and 28, I am getting two random values, multiplying them, and depositing that value into the buffer if the index is > 2 (as the buffer goes from buffer[0], buffer[1], and buffer[2]. So, we are writing a random product into memory outside of the range (or bounds) of the buffer – hence a bigger problem than just reading as we are actually writing to memory we don’t own.

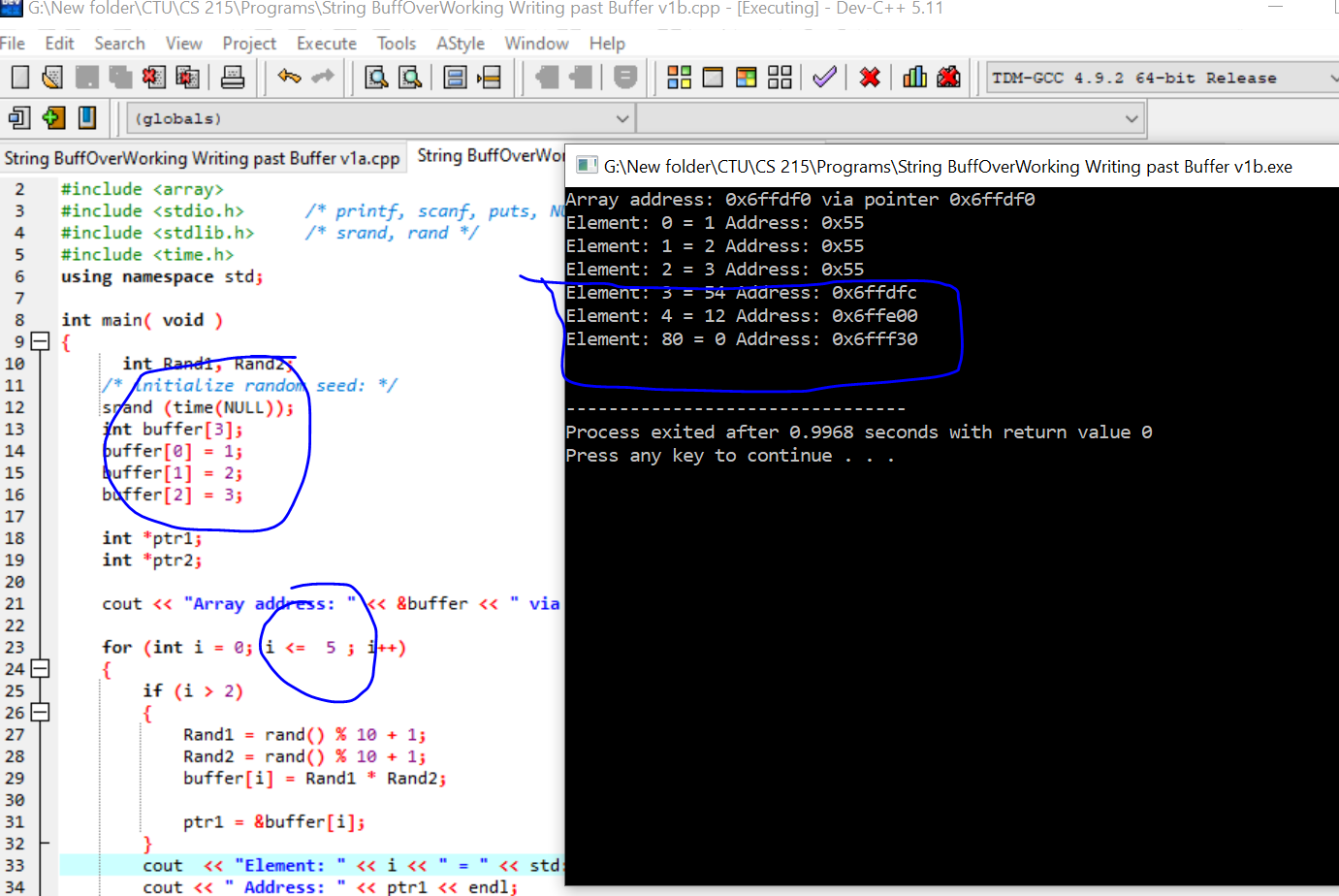


Figure 1: Buffer Overflow: Writing past the buffer