Alex Hoffer
CS 162

Reflections – Assignment 1

a) matrix.cpp:

The thing I like about this program is how smooth the output code looks. I considered also putting in vertical and horizontal lines to more effectively demonstrate the borders between the rows and columns. What I wish I could've done is been able to use pointers-to-pointers (or some other strategy) to create general createMatrices and print functions that could operate without knowing immediately the dimensions, rather than requiring that they be passed exactly in a specific order and containing a specific number of elements in each dimension. I guess I felt like I was a bit limited in this program because of the fact I could only use [2][3] and [3][2], but I haven't been able to fully comprehend how I would create a function that accepted any numbers of elements in dimensions that were compatible for each other, or else throw an error or something. I also couldn't help but wonder if my quantity of loops could be cut down.

b) war.cpp

I think this is my best out of the three. It is well modularized (main is used as a middleman between the objects and their functions) and thus well organized. I utilized two things that I read about in the book that we haven't covered yet: static member variables and operator overloading. I made the number of rounds a static member variable because it should be the same number for both players and it would be a hassle to make it non-static and have to increment both objects' round number. I overloaded ++ for convenience so I could just list the object that won the round and then increment its wins. Since the program involved randomized numbers, it was relatively easy to test: for small round numbers I printed out each number that was generated, and for large numbers (like 500 rounds) it was pretty obvious that there was randomization because I could enter 500 rounds 5 times in a row and each time both players would have different numbers of wins. I produced random numbers by including time, which starts counting seconds after January 1, 1970, and initializing each object with a different seed value using this.

c) buttercup.cpp

There are two non-main functions in this program. One receives a number corresponding to how many suitors are standing in line and then fills a vector with ints leading up to and including that number. For example, if it receives 5, the vector operates as follows: “push back 1…(check loop condition)…push back 2…(check condition)…” and stops after pushing back 5. The second function pertains to the meat of the problem: it is designed to calculate each third value that arises and then chop it off the vector. I first tried to solve the problem by using a loop that incremented by 2 each time and erased the first element + this. The problem was that since the size of the vector was changing with each erase, the loop would end up erasing a value that was past the vector. Then, I tried reading particularly into the erase function: it returns, apparently, an iterator that corresponds to the next element in the vector. I could have probably done something with that information but I felt that I would be penalized for using iterators since they haven’t been covered (I used other stuff that wasn’t covered too but this was an important part of the program. The other stuff I used was innocent, like the ++ overload). So, I thought about how I could do a loop where both the conditions of the loop itself and the counter that was being used acknowledged the changing size of the vector. I realized it was crucial that the way the counter was to be calculated needed to know that the size was different. So, I tried out using the modulus operator initially because I know that for as long as $x < y$ in the formula $x \% y$, the result is equal to x. This would allow me to perform the first calculations: if size = 5, and x = 2, then it will erase 2 % 5 which is position 2 in the vector. By adding 2 to the counter and then making counter the same as counter % size, it moves the counter forward and THEN adjusts the counter to the size. The testing process for this function was more arduous than that of the other programs in this assignment because it required I think of a number of suitors and then write out each number, count three down the line, and mark it off. I ran the same number of suitors in the program as I did on paper and compared answers. The particular number of suitors I checked for were the values 1-5 and 10. I can offer no sophisticated reasons why I chose these numbers besides 1-5 being small and 10 being just big enough. I did not check giant numbers like 500, but am confident that the result would be correct. Finally, I included a few exceptions just for fun, though they were necessary because the function is supposed to return a value.