Group 7
CSCI 313
HW1—Question 10: Even Odd Sort

Sorting is a fundamental operation in computer science, and sorting according to a given criteria is a key skill. When it comes to sorting, there are so many possible options for solutions that the possibilities can be a bit overwhelming. In order to sort a given list into odd numbers first and then even numbers, one only needs the criteria of whether x%2 is equal to 1 or to 0. All the rest is improvisation. In attempting to develop a greater familiarity with template classes and iterators, I used the class **forward_list** for a single-linked list, and used the one-directional iterator that comes with the class.

First the main function takes in a value from the user for the length of a given list. That value, k, is then used in a for loop to create a list of numbers counting up from 1 to k. Using the **push_front** function requires the loop to count down, from i = k to i > 0. Another option would be to push the first value, and then add all the remaining values using **insert_after**. Either approach works. Then, the list is passed to the function **evenOddSort** to be sorted.

The function **evenOddSort** takes in a list and creates a temporary **forward_list** it calls **temp**. It also creates two iterators, one for the received list, **it**, and another for the temporary list, **next**. Iterating through the given list, called **L**, the loop creates a value equal to ***it** and then evaluates that value to be even or odd. Using a value is not strictly necessary, ***it** should work perfectly fine, but this function threw several **nullptr** exceptions during building and so it was done out of an abundance of caution. Based on the main function, the first value of L is always 1, and therefore always odd. In the case that **val** is odd, there is a special if statement in case **temp** is empty. In that case, it pushes 1 to the front and sets **next** equal to **temp.begin().** This works to prevent next from being a **nullptr** when incremented. From then on, the loop continues using

push_front for odd values and **insert_after(next)** for even values. Every time an odd value is added to **temp**, **next** is incremented so it maintains a target on the last number in the list (**insert_after** also increments next), marking where to add the next even value. The resulting **temp** is returned and assigned to the list.

This is admittedly a somewhat ungainly sort. There a number of elegant ways to accomplish this sort, including swapping values within the current list, but it did provide a great deal of experience in terms of iterators. The outcome of this list is actually in an unusual order, with the odd numbers in descending order due to the **push_front** function, while the following even numbers are ascending. This could be remedied with a second iterator, keeping track of the end of the odd numbers, but it was agreed that this function fulfilled the brief.