For my polynomial ADT implementation, I used a map data structure to hold onto the polynomials powers and their corresponding coefficients in descending order based on the key of the map, which in this case is the power. My Program runs each method as follows (I also have comments over each method for additional information):

The constructor method simply sets highest degree integer to zero and instantiates the object.

The read from user method calls another private method, to save rewriting code, where it asks the user to input a number for the coefficient and power and checks if they are in correct format by using a regular expression function and then maps the two numbers. After receiving correct input it continues to ask for more variables until the user decides to end.

The degree method simply returns the highest degree variable which should currently be set to the highest degree in the polynomial, or 0 if there is no higher degree than 0.

The coefficient method returns either the coefficient corresponding to the power given to the method, or returns 0. Maps are strange in that without an if statement to check if the map has the given key and just returning with map[givenKey] the map will map the new key to 0. To avoid this I had to create an if statement.

The change coefficient method changes the current coefficient of the specified power to the new coefficient. The if statement just checks to see if the coefficient exists.

The multiply poly method uses a for loop to iterate through the map and multiplies each value by the given integer constant.

I decided to overload the + operator. The unfortunate side effect, which I haven’t found a work around yet, is that I cannot declare const on the second polynomial because the map won’t let me access its key value without me doing a whole lot of extra coding. I use a sort of merge sort method of combining the two maps by adding the values of identical keys and appending the keys to the new map in descending order.

The print method uses a for loop to iterate through the polynomial adding ‘x^’ and ‘+’ to create a nicely formatted polynomial. It also sets all coefficients with powers of 0 to just the coefficient seeing as anything raised to 0 is 1.

The overloaded negation operator uses a for loop to iterate through the polynomial and changes the value of each key to its negative.

My main function just runs through each possible method and prints how the specific polynomial is affected by the method.