Evan Heaton
CS315 - Homework2
9/15/15

1. For this problem, I used a recursive binary_to_decimal function. The method involved multiplying by two (in decimal) so that was a helper function I needed to create as well. Probably not the fastest and definitely not the most elegant solution, but it works.

   The test case file for this problem is named problem1.txt
   To input new test cases, type binary bits separated by spaces and delimited by a non-numerical character. The binary bits should be the reverse of the number you are trying to enter.
   *For example, to send $10110_2$ type 0 1 1 0 1 #*
   *Separate test cases with a newline*

2. This problem took a bit more work. Originally I had planned to use a recursive function to convert from decimal to binary, but this proved to be much more difficult than I had anticipated. The recursive method involved making a binary_multiply_by_10 function, which itself required a binary addition function. Yikes.
   After ripping my hair out for over an hour trying to get that to work I ended up writing an algorithm that took a decimal number and repeatedly subtracted the largest power of 2 which is much more elegant. You can see this result in my function decimal_to_binary. My final solution takes the digits of a base $2^k$ number and converts them to binary (each of these binary vectors has k bits). It then joins them into one binary vector.

   The test case file for this problem is named problem2.txt
   *To input test cases, type binary bits separated by spaces and delimited by a non-numerical character. The binary bits should be the reverse of the number you are trying to enter. On the next line, type k where the base is in the form $2^k$. Separate test cases with a newline.*

3. This was the easiest of the problems. The process here is to have a carry variable that gets set to 1 if any addition is larger than $2^k$-1, and 0 otherwise. Then at the end you can't forget to add another 1 if the carry isn't 0.

   The test case file for this problem is named problem3.txt
   *To input test cases, type binary bits separated by spaces and delimited by a non-numerical character. The binary bits should be the reverse of the number you are trying to enter. On the next line, type another string of binary bits separated by spaces and delimited by a non-numerical character. On the next line, type k where the base is in the form $2^k$. Separate test cases with a newline.*