README.pdf
Josephine Lee
leejosephines@wustl.edu
CSE241 lab 2

Part 1:
Initially, I ensured that m = maxSize is prime.
For calculating h2, I used `int h2 = (int) Math.ceil((m-1)*(k*A2-Math.floor(k*A2)));`
Multiplying `(k*A2-Math.floor(k*A2))` by `(m-1)`gives a double between 0 and m-1, not inclusive.
The outer `Math.ceil` operation then gives an integer between 1 and m-1, inclusive.

These two aspects (multiplying by `(m-1)`instead of `m`, and using Math.ceil instead of Math.floor) were necessary to avoid having `h2` equal 0 or m, which would prevent my methods from checking the entire table.

Later when I implemented the dynamically growing table that grew by powers of two this boundary was no longer needed, and I just needed to ensure that h2 is odd. I added this check that h2 is odd. However I left the h2 formula untouched because it worked like that and I didn't want to risk breaking anything.

Part 2:
To also store the toHashValue of a Record, I added an instance variable and a method to modify it in the Record class.
Having the same hash values does not necessarily mean that the strings are the same, as there may be rare cases when two strings generate the same hash value. Thus, it is still necessary to check that the strings are the same due to these rare cases.

To implement the doubling procedure, I added the instance variable `n` to the StringTable class, to keep track of how full the StringTable was. `n` was incremented when a record was inserted into a null slot. When a new record was to be inserted, if (`n`*4> the current table size), I called doubleSize(). This created a new table that was double the original size, re-set `n` to 0, and re-hashed all the current records into the new table.