Assignment Objectives:
- Implement hash tables and hash functions.
- Compare effectiveness of varying hash table length/hash function combinations.

Searching an unsorted list with a linear search has $O(n)$ time complexity, while inserting into that list has $O(1)$ time complexity. Searching a sorted list with a binary search has $O(\log n)$ time complexity, while inserting into that list also has $O(\log n)$ time complexity. What we'd really like is a data structure that allows us to both search and insert into a list with $O(1)$ time complexity. One structure that has been devised to help accomplish this goal is the hash table.

The main idea behind the hash table is to use the data itself to decide where to store it. A hash function is formulated to convert certain aspects of the data into a number that is then used to index the table. The data in question is then placed in the table at the indexed location. For example, if the data entries are all employee records, one hash function could use the last digit of the employee's Social Security Number to determine the table slot (numbered zero through nine) to which the employee's record would be inserted, as illustrated at right.

The two big tricks to making hashing effective are:
1. determining what to do in the event of a collision, i.e., when two pieces of data are mapped to the same slot, and
2. choosing a good hash function, which distributes the data evenly among the table's slots.

In this programming assignment, you will address both of these issues. To handle collisions, you'll use open hashing, which merely places all data with the same hash value in a linked list in that slot of the table. Thus, in our example at right, inserting two additional employees, with Social Security Numbers 282-06-2135 and 537-54-7845, would result in a modified Slot #5 as illustrated below:

![Hash Table Diagram]

To address the issue of whether or not a hash function is good, we're just going to try a few different hash functions on several data sets, and determine how long the linked lists in the resulting hash table become, which, in turn, reflects the time it takes to search or insert into the hash table. To bring hashing as close as possible to the $O(1)$ time complexity goal that we've set, we'll need the data being mapped to be distributed as evenly as possible over the linked lists in the hash table (i.e., the longest linked list in the table shouldn't vary that much in length from the average length of these lists).

The data you'll be using are simple phone directories containing numerous phone-number/name pairs. You'll use seven different hash functions, which merely add one to each of the last four numerical digits in each phone number, multiply then together, and apply a modulo function (% in C++) to the product. The modulo values applied for the seven hash functions will be 17, 37, 67, 131, 257, 521, and 1031, each resulting in a hash table of the corresponding size. Thus, for example, the modulo-37 function would produce a 37-slot hash table, with each slot containing a linked list of phone-number/name pairs. The pair (555-6762, John Smith) would map to slot #29 (since $7 \times 8 \times 7 \times 3 = 1176$, which is 29 mod 37) and the pair (650-1983, Jane Jones) would map to slot #17 (since $2 \times 10 \times 9 \times 4 = 720$, which is 17 mod 37). The same pairs would have mapped to slots #148 and #206 (respectively) for the modulo 257 hash function, slots #152 and #199 (respectively) for the modulo 521 function, and slots #145 and #720 (respectively) for the modulo 1031 function.
After filling each hash table, your program should output the following statistics concerning the table:

- the size of the table (i.e., how many linked lists comprise the hash table),
- the average number of linked list nodes that would have to be traversed to reach a particular phone-number/name pair,
- the maximum number of linked list nodes that would have to be traversed to reach a particular phone-number/name pair (i.e., the length of the longest linked list in the hash table), and
- the length of each of the linked lists comprising the hash table.

You will need a driver program to read the phonebook data from an external file, set up the seven hash tables, and analyze and output the results. In addition, three classes must be implemented:

1. A `linked list` class template (using the version presented in class or the version presented in the Weiss text is acceptable, but you might find it useful to add some functionality),
2. A `hash datum` class, to hold the string data members for the name and phone number (member functions should include constructors, accessor and mutator functions, an input operator, and a function to multiply the increments of the phone number digits), and
3. A `hash table` class, to hold a `dynamic` array of hash datum linked lists (member functions should include constructors, accessor functions, a subscript operator, a function to insert a hash datum, and functions to perform the average and maximum traversal counts).

The output below demonstrates the format that your output should take. Three sample input files are available on the course website. You may assume that all input files for this program are formatted in this manner.

```
<table>
<thead>
<tr>
<th>TABLE SIZE</th>
<th>AVERAGE TRAVERSAL</th>
<th>MAXIMUM TRAVERSAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>1.43</td>
<td>4</td>
</tr>
<tr>
<td>127</td>
<td>2.29</td>
<td>12</td>
</tr>
<tr>
<td>67</td>
<td>3.38</td>
<td>6</td>
</tr>
<tr>
<td>133</td>
<td>1.90</td>
<td>6</td>
</tr>
<tr>
<td>257</td>
<td>1.77</td>
<td>6</td>
</tr>
<tr>
<td>321</td>
<td>1.77</td>
<td>6</td>
</tr>
<tr>
<td>1811</td>
<td>1.77</td>
<td>6</td>
</tr>
</tbody>
</table>
```

Zip-compress your entire program folder and place it on your dropbox by the deadline mentioned above. Several files of test data are available on the course website.

You must write your own code on this assignment, with adequate explanatory documentation (i.e., a paragraph at the top of each program file, including your name and a description of the file’s contents, and a sentence preceding each function, describing the function’s purpose).

Obtaining code assistance from any outside source is considered academic misconduct.

The only person permitted to see your code prior to the assignment deadline is the instructor.