These must be completed and shown to your lab TA either by the end of this lab, or by the start of your next lab. You can work in groups of up to two people.

1. Download the hash table code from the course webpage under Lab 7. You will be comparing three different open addressing schemes.
2. Complete the following functions in hashtable.cpp:

```
// Insert k in the hash table.
// Use open addressing with quadratic probing and hash(k) = k % m.
void Hash::qinsert(int k) { }

// Insert k in the hash table.
// Use open addressing with linear probing and hash(k) = k % m.
void Hash::linsert(int k) { }

// Insert k in the hash table.
// Use open addressing with double hashing. Use the existing hash function
// and also implement a second hash function
void Hash::dinsert(int k) { }
```

   You must complete these functions so that they call `tallyProbes` on every successful insertion, with the number of probes required for that insertion. Note that:

   ```
   ./hash q 900000 1000000
   ```

   will insert 900,000 random keys into a hash table of size 1,000,000 using quadratic probing, while

   ```
   ./hash d 8000 10000
   ```

   will insert 8,000 random keys into a hash table of size 10,000. Replacing `d` with `l` will insert keys using linear probing.

3. Run a few experiments to see how the average number of probes per insertion differs when using the different collision resolution methods and different load factors. You should think of this as a regular scientific experiment that generates data. You need to run enough experiments to justify your conclusions, and you need to use a table large enough. Try to make the table as large as possible such that you do not have to wait an unreasonable amount of time for the results. Answer the following questions:
   i. Under what load factors is linear probing just as good as quadratic probing? When does quadratic probing begin to perform better?
   ii. How does the choice of hash function affect double hashing? Can you devise a hash function that makes double hashing just as good as quadratic probing? Can you make it even better? Make sure you try a few different secondary hash functions to be able to justify your answer.
   iii. When does double hashing begin to win against both of the other schemes? (i.e. at what capacities and load factors do you see a significant gain over the other methods?)
4. Modify the main method in lab7driver.cpp to automate some of your experiments from question 3. Be prepared to explain what you tested to your TA. You can run your added experiment code using:

```
./hash
```

For example, you could add something like this:

```
Hashtable H(10000);
for (int i = 0; i < 7500; i++) {
linsert(rand() + 1);
}
cout << "Linear with 7500 keys and size 10000: ";
H.printStats();
```

to automatically see how insert with linear programming performs when inserting 7,500 keys into a hash table of size 10,000. What other experiments might you automate to help you answer the questions from 3?

5. Be sure to show your work to your TA before you leave, or at the start of the next lab, or you will not receive credit for the lab!