As with previous chapters, we are going to lay out the learning objectives for this chapter, and split them into what you should get on the first reading vs the second reading.

1 First Reading: Understand Key Ideas

On your first reading, the goal is to get the big ideas, and build the mental scaffolding in which to hold the details on your second reading—the learning objectives for your first reading are generally **Remember** and **Understand** objectives. If you are missing a few details (but not completely lost), it is ok to move ahead, and come back to them the next time through—you might ask questions on the course forums too! Note you should **sleep at least one night between the first and second reading** to give your brain time to process the information.

- **Hash Table Basics**
  - **Define** *hashing function.*
  - **Explain** the structure and basic operation of a hash table.
  - **Explain** why inserting or finding a particular key in a hash table is $O(1)$.
  - **Explain** how we can make a hash table generic in its key type.
- **Hash Table Basics**
  - **Define** *collision.*
  - **Define** *chaining.*
  - **Explain** how hash tables resolve collisions with chaining.
  - **Define** *buckets.*
  - **Explain** why we still expect access to a hash table to be $O(1)$ with chaining.
  - **Explain** why we use linked lists, not binary search trees to chain a hash table.
  - **Recognize** that chaining is the most common general-purpose collision resolution strategy.
  - **Define** *open addressing.*
  - **Define** *linear probing.*
  - **Define** *quadratic probing.*
  - **Explain** how to add to a hash table using open addressing.
  - **Explain** how to find an item in a hash table using open addressing, and why this is more complicated if the hash table allows removing data.
- **Hashing Functions**
  - **List** and **explain** the two criteria for a function to be a valid hashing function.
  - **Explain** what makes a valid hash function good or bad.
  - **Explain** why just summing the letters in a string is a bad hash function.
  - **Explain** why multiplying the current sum by a constant before adding in the next letter in the string makes a huge improvement in how good the hash function is for strings.

- **Recognize** that characteristics of functions depend on input data (not just hash functions: this LO is a good general lesson).
- **List** and **explain** three guidelines to follow whenever you need to write your own hash functions.
- **Recognize** that hashing functions as we have been discussing are different from cryptographic hash functions.
- **Recognize** that storing passwords in plain-text is an unacceptably bad practice from a security standpoint.
- **Define** *salt*.
- **Explain** why `hash(password+salt)` is the best practice for storing passwords (using a cryptographic hash function).
- Note: this is not a security class, but we do want to underscore this security practice as it is an easy thing to do and a big deal. We recommend a security class for in-depth coverage of security topics.
- **Explain** how to implement a hash function with the overloaded `()` operator.
- **Recognize** that `std::hash<T>` provides a hashing function (via the overloaded `()` operator) for some built-in types (including `std::string`). Note: you can provide your own specialization for `std::hash`, though we don't cover explicit specializations.

- **Rehashing**
  - **Define** *rehashing*.
  - **Define** *load factor*.
  - **Explain** how the load factor is used to determine when rehashing is needed.
  - **Explain** how to rehash a hash table.
  - **Explain** what advantages arise from keeping the size of a hash table as a power of 2.
  - **Explain** what advantages arise from keeping the size of a hash table as a prime number.
  - Note: the nasty mathematical details are way outside the scope of this class, and are left to theoreticians—you don't even really need them in a theory class.

2 Second Reading

Before you do your second reading, we just want to remind you that we have lighter coverage of this chapter than the others. Hash tables are still great things to know about, and especially to use ($O(1)$ maps and sets—yay!). You'll get less practice with hash tables than with other ideas in this course. However, you should still aim for the following on your second reading (after a good night's sleep!)

- **Show** how items are added to a hash table with chaining or open addressing.
- **Implement** a hash table (note: once you understand the algorithms, implementing a hash table is just a matter of applying your vector and linked list skills).

- **Assess** if a hash table is appropriate to for the problem you are solving (note: they are great for map and set ADTS, whenever you only need equality—if you need to efficiently find ranges, or things that are less/greater, a BST might be a better choice).
- **Write** code that uses a hash table.