Capacity And Load Factor, Reading

We have not paid much attention to capacity. We've just been assuming that the capacity will always be sufficient. In a template with a dynamically allocated data array, there would have to be some arbitrary trigger for doubling, because you don't want to wait for the array to be full before doing something. And when you *do* change capacity, you cannot just copy Nodes to their same old indexes -- that all have to be removed, queued up, and reinserted, just like in rehashing.

The issue is avoided with a static array, but the programmer has to make sure the capacity sizing is appropriate for the application. But in either case we need some way to know "how big is big enough" for the data array.

Load Factor

The "load factor" is a measure of how full is the data array. It's calculated as the **"siz" divided by the capacity**. Obviously the hash table with probing would not work at 100% load, because there's no place to put new additions. Near 100%, it's about O(n) to find one of the few unused indexes, or to search for a matching key. But at 50%, the chance of the calculated wrapped index being available is 0.5. The chance of it being occupied but the *next* one vacant is 0.25, and it drops by half for each successive. The average number of cycles in a search loop is then 1x0.5 + 2x0.25 + 3x0.125 +... a mathematical series that sums to 2. At 60% load, this rises to 2.5. At 70%, 3.3, and it goes up rapidly from there. A 50% load factor maintains O(1) but at a cost of a constant multiplier of 2 -- not too bad, and a good "rule of thumb" is to size the data array for a maximum load of **50% for probing**.

For chaining the math is a bit different -- it involves "order statistics". There's no probing. There's just searching of lists. At a load factor of 50% you might have 10% of the lists with 2 values and 30% with 1, the rest empty. That's a constant multiplier of 1.1 -- better than probing. The load factor has to get up to 80% before you'd get close to a multiplier of 2. A good rule of thumb is to size the data array for a maximum load of **80% for chaining**.

A Load Factor Getter

So for the programmer to be able to monitor the load factor in an application using a static data array, or for the template to monitor load factor itself when using an adjustable dynamic array, it's good to have a public getter, which can easily be written inline:

```
double loadFactor() const {return 1.0 * siz / CAP;}
```

The 1.0 is so that the division operation will not be int/int, which truncates any fractional part. We could have done a static cast of one or both of the ints instead.