

## Exploring the unordered\_map

In this exercise, we'll spend a little time understanding how the STL implements hash tables. We've already seen that the `unordered_map` uses something like chaining (though whether the "buckets" are "chains" isn't totally clear from the documentation). But how many buckets are there? How big are they? And so on.

### Understanding the Basics

Write a small program as follows:

Create a "basic" `unordered_map` where `key_type` and `mapped_type` are both `int`. (That is, we'll just store ints in the hash table for now.)

Use the `bucket_count` and `load_factor` methods to learn about the default initial state of an `unordered_map`.

Add 10 unique values to the map. After each addition, check the `bucket_count` and `load_factor`. Run the program, and based on these results, make a hypothesis about how the map is resized. According to your hypothesis, what should the `bucket_count` and `load_factor` be after (say) 100 values have been added to the map?

Now add to your program—add 90 more values, so there are a total of 100 values in the map. Output the `bucket_count` and `load_factor`; also use `bucket_size` to find out the max number of elements stored in any one bucket. How does that number relate to the `load_factor`?

### The Danger of Bad Hash

OK, let's implement the `Point` class, along with a bad hash function; let's see what that does to our maps. The `Point` class can be super-simple:

```
class Point {
private:
    int x, y;
public:
    int getX() return x;
    int get&() return y
}

std::ostream& operator<< (std::ostream& out, const Point& p) {
    out << "(" << p.x << ", " << p.y << ")";
}
```

But we also need to implement a hash function that will work with `unordered_map`. To do that, we have to specialize the `std::hash` template to work for `Point`. We'll use the bad hash function `x+y`; to see how to incorporate that, look at the example code at [http://en.cppreference.com/w/cpp/utility/hash/operator\(\)](http://en.cppreference.com/w/cpp/utility/hash/operator()).

Now write a main program that creates a hash table for `Points`. Add 100 points to the table, from (0,0) to (9,9). What are the `bucket_count` and `load_factor` for this table? Are they the same as for the previous table of 100 ints? Does the "bad hash function" seem to be affecting these values?

Follow the example at [http://www.cplusplus.com/reference/unordered\\_map/unordered\\_map/bucket\\_count/](http://www.cplusplus.com/reference/unordered_map/unordered_map/bucket_count/) to print the contents of each bucket. Does the "bad hash function" seem to be affecting these results?