Maps

Pair Type (#include<utility>)

- C++ provides a pair type
  - Holds **exactly** two values
  - Is templated on the two values
- `pair<string, int> word_count;`
- single element with two parts, a string and an integer

Members and functions

- `make_pair("hi mom", 12);`
  - Returns a `pair<string, int>` type
    - Types inferred by the compiler
- `pair<string, int> wc = {"hi mom", 12};`
  - Make one and assign
- `wc.first` or `wc.second`
  - first element or second element
  - **Not a function**
  - A data member

Which of is the type of x?

```
auto x = make_pair(3, "happy").second;
```

- It would not compile
- `string`
- `int`
- I don't know

Can’t print a pair

- Much like a vector or any compound pair, you cannot print a pair:
  - You have to print the elements
  - Algorithms are your friends!

Ordered Associative Containers

- map
- multimap
- set
- multiset

maps are not a sequence

- It is important to remember that a map is not a sequence
- Maps have an ordering, but it is not the order that the elements were inserted into the map

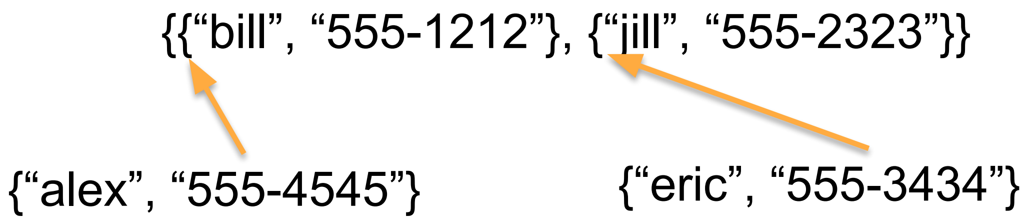
Bidirectional iterators

- These containers yield bidirectional iterators (not a sequence remember)
  - Can advance iterator both forward and backward
  - No random access via `[]`
  - No pointer arithmetic
  - No `itr < v.end()`
  - Does allow `itr != v.end()`

Ordered containers: map

- `map` automatically inserts new elements such that they are ordered:
  - Each `map` element is a **`pair`**
    - (key, value) in that order
  - Order of map elements is based on the key
  - If not specified, the order is based on a less-than compare on keys
  - Search for elements is very fast

Maintains order of keys



```
{{“bill”, “555-1212”}, {“jill”, “555-2323”}}
{“alex”, “555-4545”}
{“eric”, “555-3434”}
```

Initialization and Keys

```
map<string, string> authors = {
     {“Joyce”, “James”}, {“Austen”, “Jane”},
     {“Dickens”, “Charles”}
     };
```

- Directly indicate the pairs
- Only requirement on keys is that they must have a way to compare keys (these containers are ordered)
- Either by default or you provide one

By Iteration

```
using Cnt = pair<char, long>;
vector<Cnt> v = {{‘a’, 0}, {‘b’, 1}};
map<char, long> m(v.begin(), v.end());
```

- Push back pairs (of the correct type) onto the map

map, 3 ways to insert

- Not `push_back`, rather `insert`

```
map<string, int> m;
string word = “hello”;
m.insert({word, 1});
m.insert(make_pair(word, 1));
m.insert(pair<string, int>(word, 1));
```

Much like a Python dict

- Every key has an associated value
- Fast search is by key to find that value
  - Cannot do the reverse, find value and look up key

map, return from insert

- `insert` returns a `pair<iterator, bool>`
  - If key is in map, then insert does nothing and the second element of the returned pair is false
  - If key is not in the map, the insert works and the second element of the returned pair is true
  - Iterator points to element (whether added or already there)

3 ways to erase

```
map<string, int> m;
size_t num;
// removes **every** example of key
// returns how many erased
num = m.erase(key);
```

[] operator

- Like Python, the type in the `[]` is the key and the value is what is associated (what is returned and can be assigned).
- Unlike Python, `[]` operator allows for **non-existing** keys. Any reference to a key that doesn't exist creates the key with the **default value type**

```
map<int, double> m;
++m[15];  // default double is 0, add 1
```

More Map Methods

Iteration

```
map<string, int> word_dict;
word_dict[“bill”] = 10;
++word_dict[“fred”]; // ?
for (auto itr = word_dict.begin();
    itr != word_dict.end();
    ++itr)
  cout << itr->first << endl; // ?
```

What does -> mean?

- What you iterate through in a map are **pairs**
- A map iterator points to a **pair**
- If you want to print the key of the pair via the iterator, you could type
    - `(*itr).first;`
    - `itr->first;`
- The `->` operator means a member of what the iterator points to

Cannot change a key

- Iteration is through pairs and the key is a const value
  - You can view but cannot change a key value via iteration!

```
map<int, int> pt = {{2, 2}, {4, 4}};
for (auto itr = pt.begin(); itr != pt.end(); ++itr) {
  itr->second = itr->second + 2;
  // itr->first = itr->first + 4; // error
}
```

Count words example

- Pretty straight forward to print in word order
- Printing in occurrence order is a little work

find

- Can’t uses `[]` to check for a value because it adds it if it is not there
  - `m.find(key)` // itr to key (or end)
  - `m.count(key)` // occurrences (1)

Can provide a compare function

- Ordered map (all the ordered types) maintain an order of pair elements based on keys
  - Default is less_than
- You can provide your own function and change the order
  - Easier when we teach how to make custom classes

Sets

Sets

- Sets represent mathematical sets
  - Are templated for one type
  - Hold only one example of any element
    - If you add a duplicate of an existing element, it is ignored

Can a set contain both 0 and 0.0?

- Yes, they are different types
- No, they are different types
- No, they compare equal to each other
- I don't know

Insert/Erase are the similar to map's

- Insert on a set returns a pair, just like before
  - Now the iterator points to the base type, not a pair
  - erase erases all examples of the key
    - Only one…

Iterators on sets are const

- You can iterate through a set, but the iterator is const
  - Cannot change a key in place

Set-like algorithms

- Interestingly, there are no methods for sets like union, intersection, etc.
- Instead, there are generic algorithms which can be used on any container to get that kind of behavior
- For the algorithms to work they must be working with a **sorted container**
  - Weird / undefined behavior if not already sorted

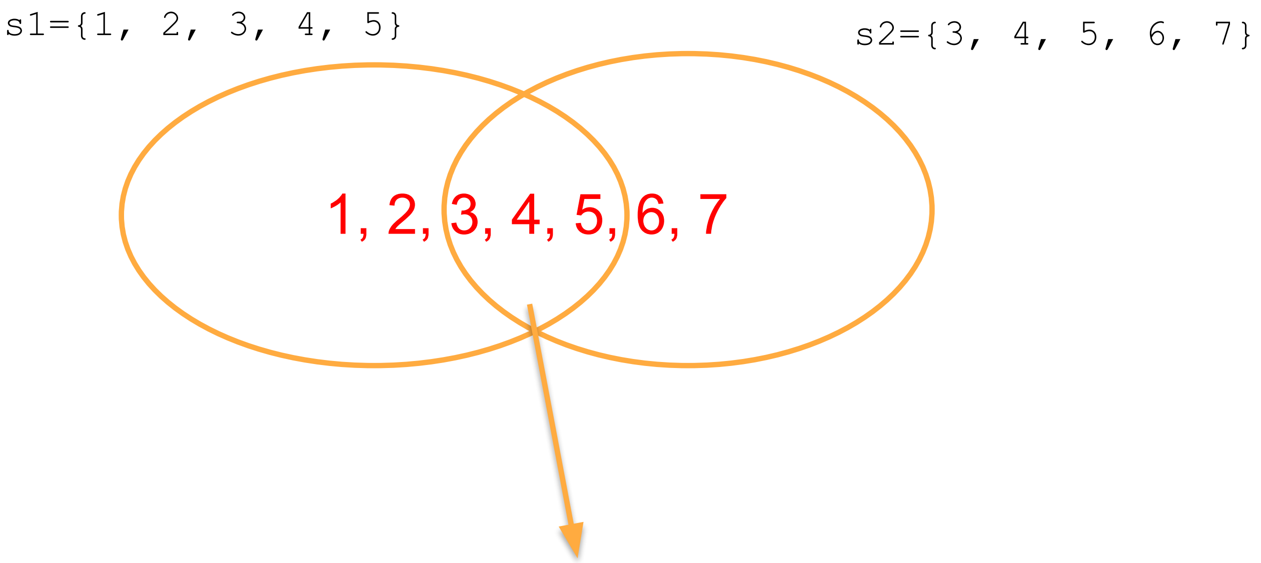
Set algorithms

- General form:
  - algorithm(src1-iter, src1-iter, src2-iter, src2-iter, dest-iter);
- Assumption is src1 and src2 are sorted, dest-iter is either another container or an output iterator

Set Algorithms

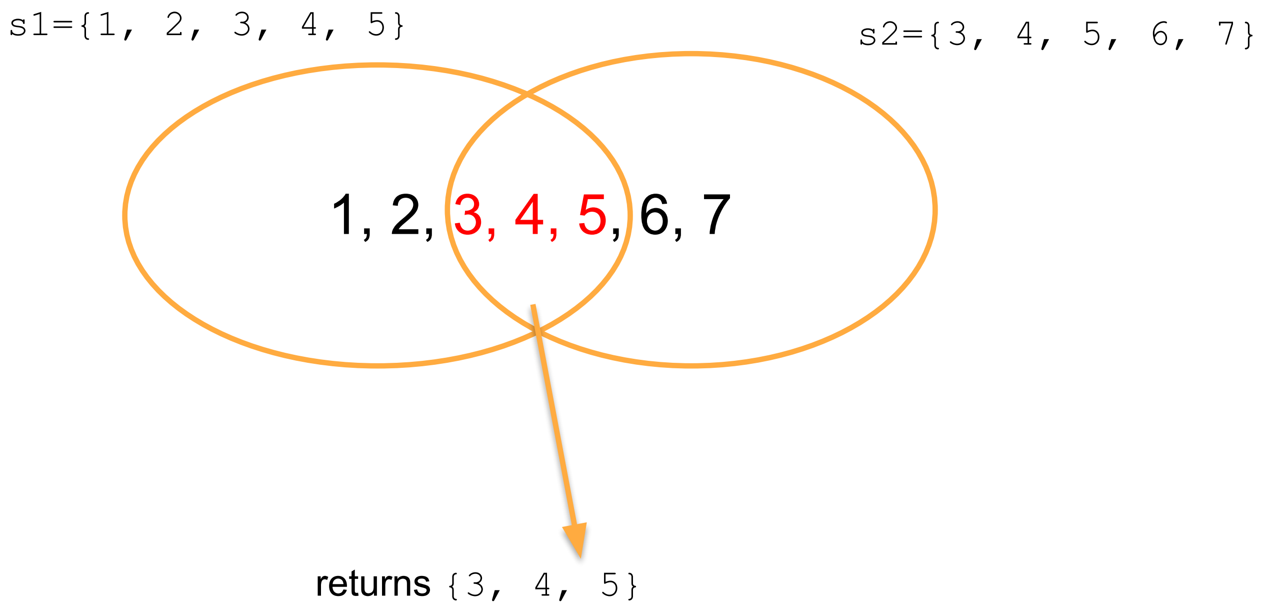
- `set_union`
- `set_intersection`
- `set_difference`
  - Those things in src1 not found in src 2
  - Order dependent!
- `set_symmetric_difference`
  - Those things found in src1 and src2 that are not common between them

Set union

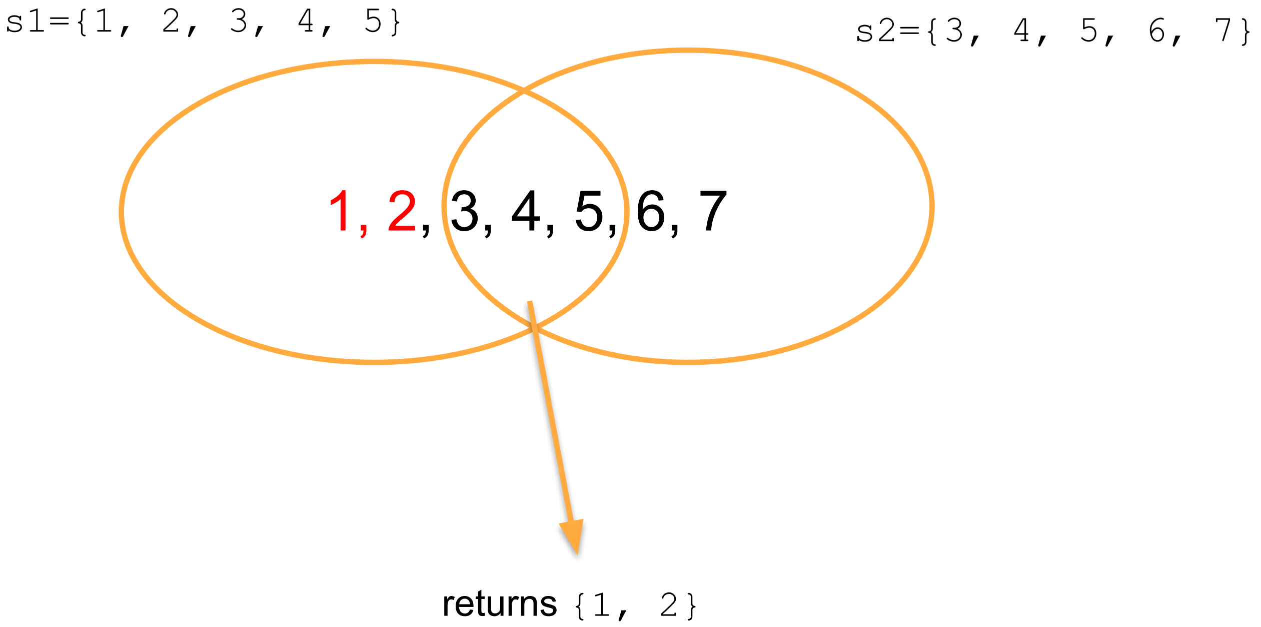


returns {1, 2, 3, 4, 5, 6, 7}

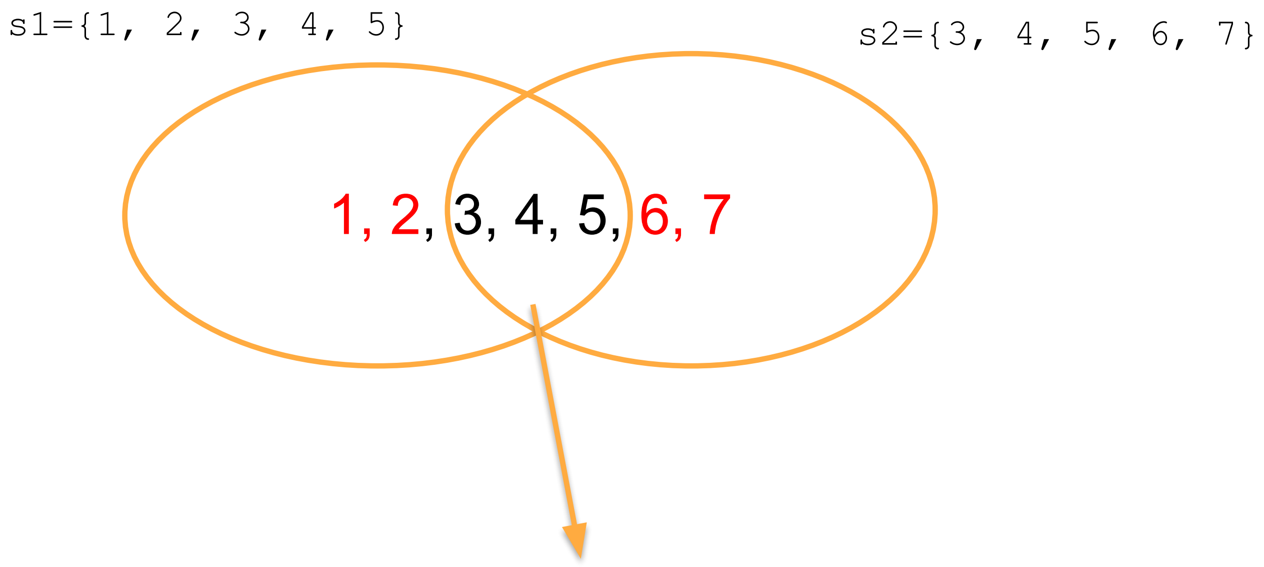
Set intersection



Set difference (s1 – s2, order matters)



Set symmetric difference



returns {1, 2, 6, 7}

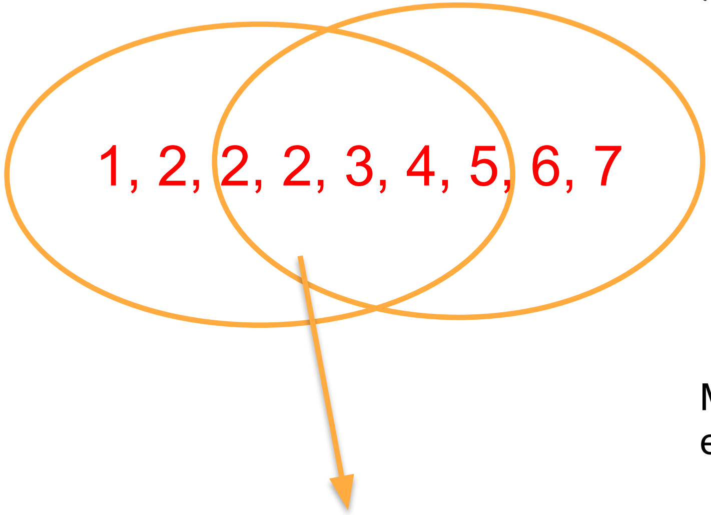
What about repeats?

- These algorithms work on any STL container.
- What happens with repeats?
- Remember, if you want to hold onto repeats, you need to insert them into a container that allows repeats

Set union

```
v1={1, 2, 2, 2, 3, 4, 5}
```

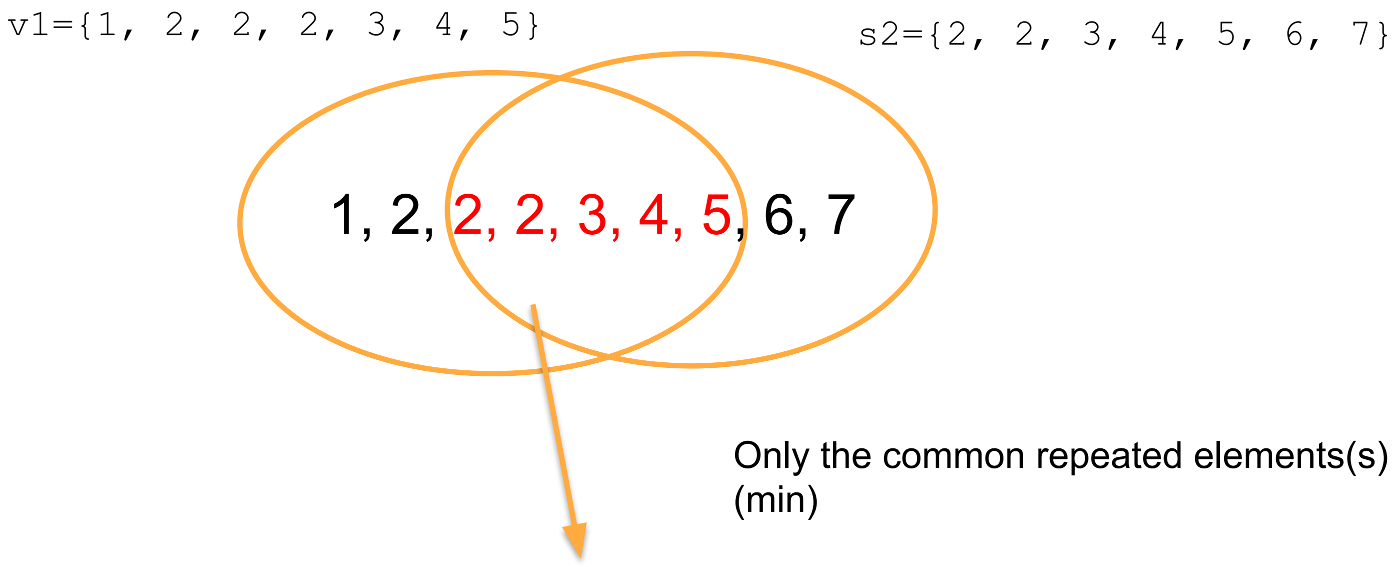
```
s2={2, 2, 3, 4, 5, 6, 7}
```



Max of the repeated elements

returns `{1, 2, 2, 2, 3, 4, 5, 6, 7}`

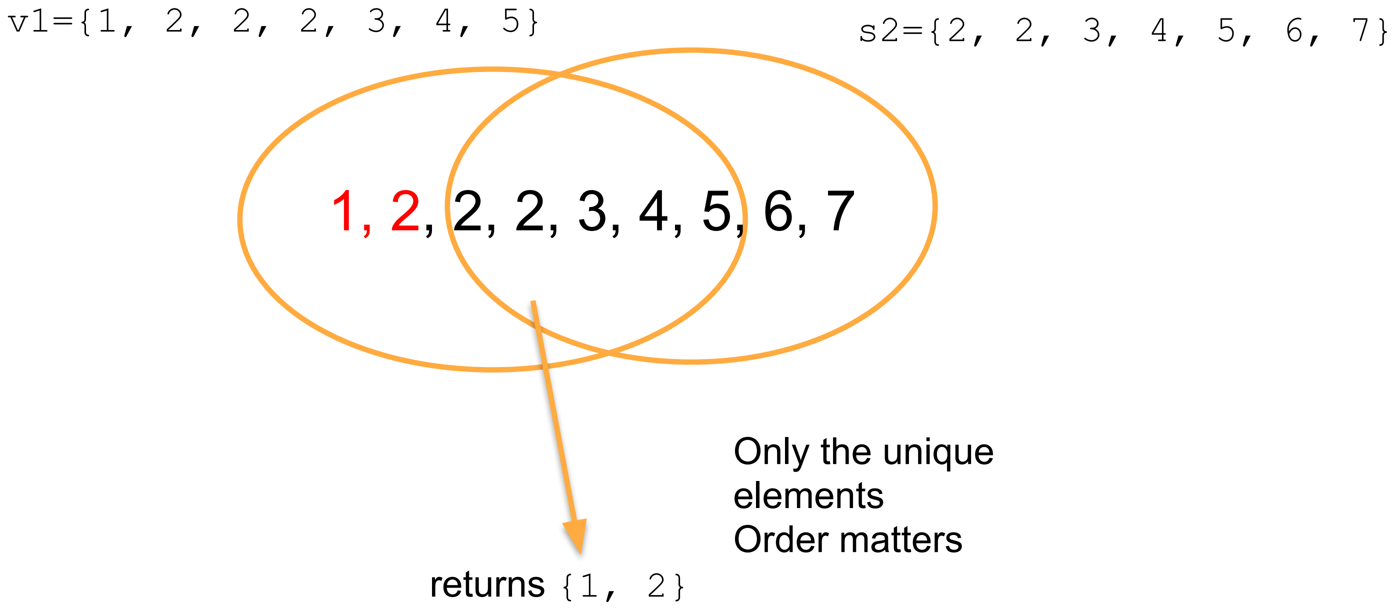
Set Intersection



Only the common repeated elements(s) (min)

returns `{2, 2, 3, 4, 5}`

Set Difference

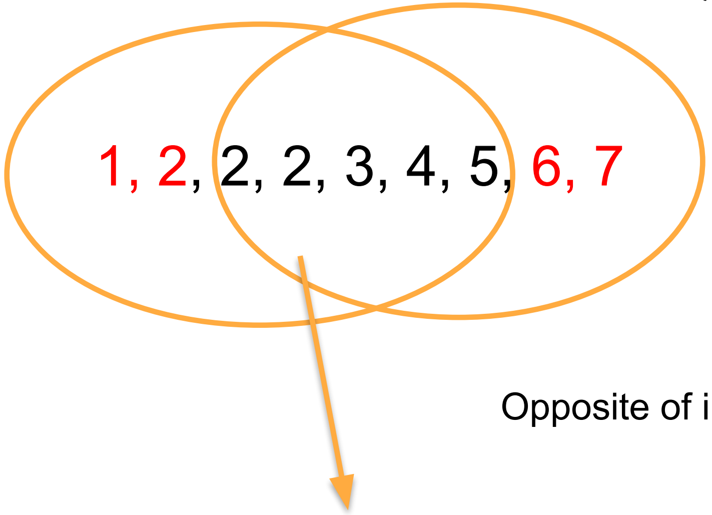


Only the unique elements
Order matters

Set Symmetric Difference

`v1={1, 2, 2, 2, 3, 4, 5}`

`s2={2, 2, 3, 4, 5, 6, 7}`



Opposite of intersection

returns `{1, 2, 6, 7}`

Multi and Unordered

Multisets / Multimap

- Multiple examples of a key are allowed
  - `multimap` is nice for “overloaded” keys (one word, multiple definitions)
  - Cannot use `[]` for either
  - `find` is useful here

More multi

- `insert` returns the iterator, not a pair
  - insert always works since multiple keys
- `count` can now return more than 1, 0
- `find` is the first element with key
  - or end if not there

unordered containers

- unordered_map
- unordered_multimap
- unordered_set
- unordered_multiset

A difference of implementation

- The unordered types do not necessarily introduce any new capabilities from the point of view of the user
- Rather than provide a new interface, they provide a new underlying implementation

Order vs Hashing

- If the elements of a container are ordered, search for an element is very fast
  - Binary search
- Another approach is called hashing
  - Make a key out of some processing of the value being stored
  - Allows for finding the item without searching (more or less), which is even faster than binary search
  - Items are stored in no particular order