Map and MultiMap

At the end of this topic, you will be able to perform the following with Maps and MultiMaps

Use a comparator function to order keys
Determine when to use a Map versus a MultiMap
Perform the following:

- Count
- make_pair
- insert
- initialize with an initializer list
- locate with the subscript operator

- *`multimap` associative container*
  - Used for fast storage and retrieval of keys and associated values
    - key/value pairs
- Many of the functions used with `multiset`s and `set`s are also used with `multimap`s and `map`s
- The elements of `multimap`s and `map`s are `pair`s of keys and values instead of individual values
- When inserting into a `multimap` or `map`, a `pair` object that contains the key and the value is used
- The ordering of the keys is determined by a comparator function object
  - In a `multimap` that uses integers as the key type, keys can be sorted in ascending order by ordering them with comparator function object `less< int >`.

- Duplicate keys are allowed in a `multimap`
  - Multiple values can be associated with a single key
  - one-to-many relationship
  - For example
    - In a credit-card transaction-processing system
      - One credit-card account can have many associated transactions
    - In a university
      - One student can take many courses
      - One professor can teach many students
    - In the military
      - One rank (like "private") has many people
- A `multimap` supports *bidirectional iterators*
  - But not *random-access iterators*

- Duplicate keys are allowed in a `multimap`
  - Multiple values can be associated with a single key
  - one-to-many relationship
  - For example
    - In a credit-card transaction-processing system
      - One credit-card account can have many associated transactions
    - In a university
      - One student can take many courses
      - One professor can teach many students
    - In the military
      - One rank (like "private") has many people
- A `multimap` supports *bidirectional iterators*
  - But not *random-access iterators*

- Figure 15.17 demonstrates the *`multimap` associative container*

- Header `<map>` must be included to use class `multimap`

- If the order of the keys is not important
  - You can use `unordered_multimap` (header `<unordered_map>`) instead.

Performance Tip 15.11

A `multimap` is implemented to efficiently locate all values paired with a given key.

```
multimap< int, double, less< int > > pairs;
```

- Creates a `multimap`
  - In which the key type is `int`
  - The type of a key's associated value is `double`
  - The elements are ordered in *ascending order*

```
cout << "There are currently "
     << pairs.count( 15 )
     << " pairs with key 15 in the multimap\n";
```

- Uses function count to determine the number of key-value pairs with a key of 15

```
pairs.insert( make_pair( 15, 2.7 ) );
```

- Uses function `insert` to add a new key-value pair to the `multimap`
- `make_pair ( 15, 2.7 )`
  - Creates a `pair` object in which `first` is the key (`15`) of type `int` and `second` is the value (`2.7`) of type `double`
- Automatically uses the types that were specified for the keys and values in the `multimap`'s declaration

```
cout << "After inserts, there are "
     << pairs.count( 15 )
     << " pairs with key 15\n\n";
```

- Outputs the number of pairs with key 15
-

As of C++11, you can use list initialization for `pair` objects

```
pairs.insert(make_pair(15, 2.7));
```

Can be simplified as

```
pairs.insert({15, 2.7});
```

Similarly, C++11 enables you to use list initialization to initialize an object being returned from a function

If a function returns a `pair` containing an `int` and a `double`

You could write:

```
return { 15, 2.7 };
```

```
for ( auto mapItem : pairs )
{
   cout << mapItem.first  << '\t'
        << mapItem.second << '\n';
}
```

- Outputs the contents of the `multimap`, including both keys and values
- Infer the type of the loop's control variable
  - `pair` containing an `int` key and a `double` value
  - With keyword `auto`

- If you know the key–value pairs in advance
  - You can use list initialization when you create the `multimap`

- The following statement initializes a `multimap` with three key–value pairs that are represented by the sublists in the main intializer list:

```
multimap<int, double, less<int>> pairs =
    {{10, 22.22}, {20, 9.345}, {5, 77.54}};
```

- The *`map` associative container* (from header `<map>`)
  - Performs fast storage and retrieval of *unique keys* and *associated values*
  - Duplicate keys are *not* allowed
    - A single value can be associated with each key
  - one-to-one mapping
- A company that uses unique employee numbers, such as 100, 200 and 300
  - Might have a `map` that associates employee numbers with their telephone extensions—4321, 4115 and 5217, respectively
- With a `map`
  - You specify the key
  - And get back the associated data quickly
- Providing the key in a `map`'s subscript operator `[]`
  - Locates the value associated with that key in the `map`

- Insertions and deletions can be made *anywhere* in a `map`
- If the order of the keys is not important
  - You can use `unordered_map` (header `<unordered_map>`) instead

- Figure 15.18 demonstrates a `map` and uses the same features as Fig. 15.17 to demonstrate the subscript operator.

```
pairs[25] = 9999.99;
pairs[40] = 8765.43;
```

- When the subscript is a key that is already in the `map`
  - The operator returns a reference to the associated value
- When the subscript is a key that is *not* in the `map`
  - The operator inserts the key in the `map` and returns a reference that can be used to associate a value with that key
    - creating an association

This topic taught and demonstrated how to perform the following with Maps and MultiMaps

Use a comparator function to order keys
Determine when to use a Map versus a MultiMap
Perform the following:

- Count
- make_pair
- insert
- initialize with an initializer list
- locate with the subscript operator