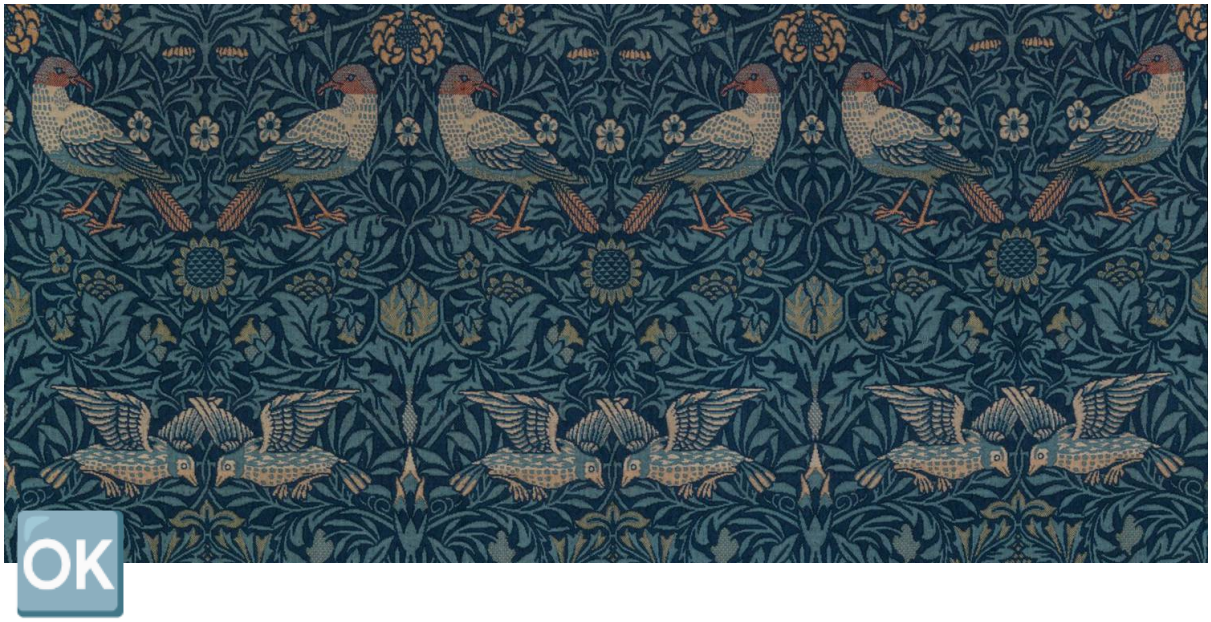


Map

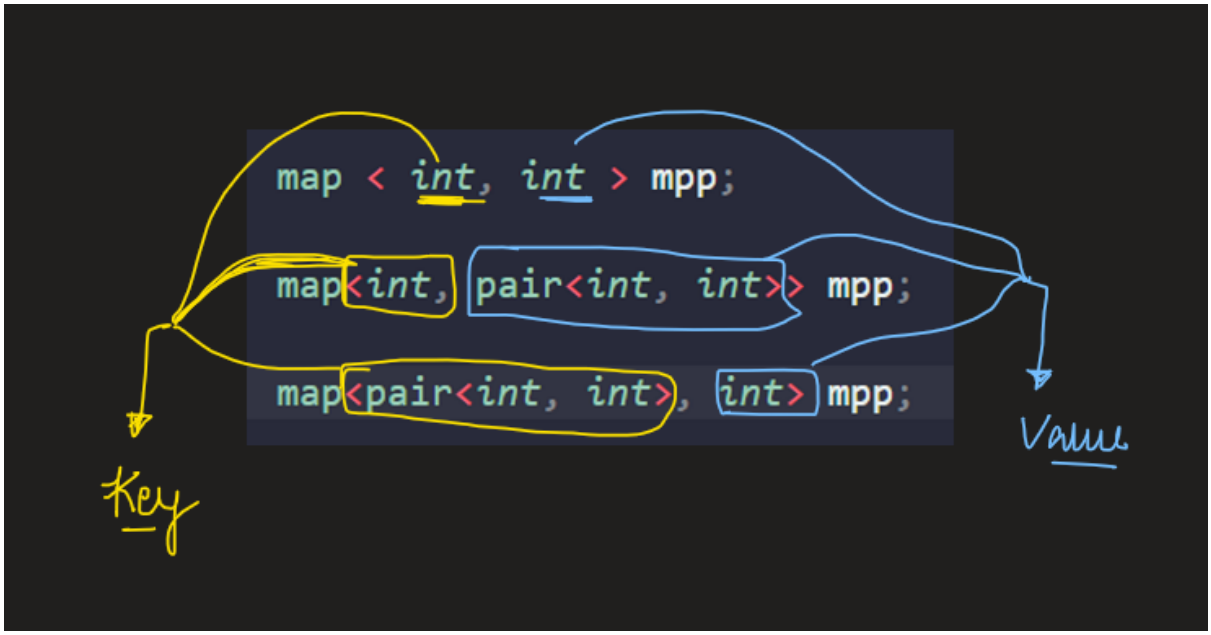
Map in STL are associative containers where each element consists of a key value and a mapped value. Two mapped values cannot have the same key value.

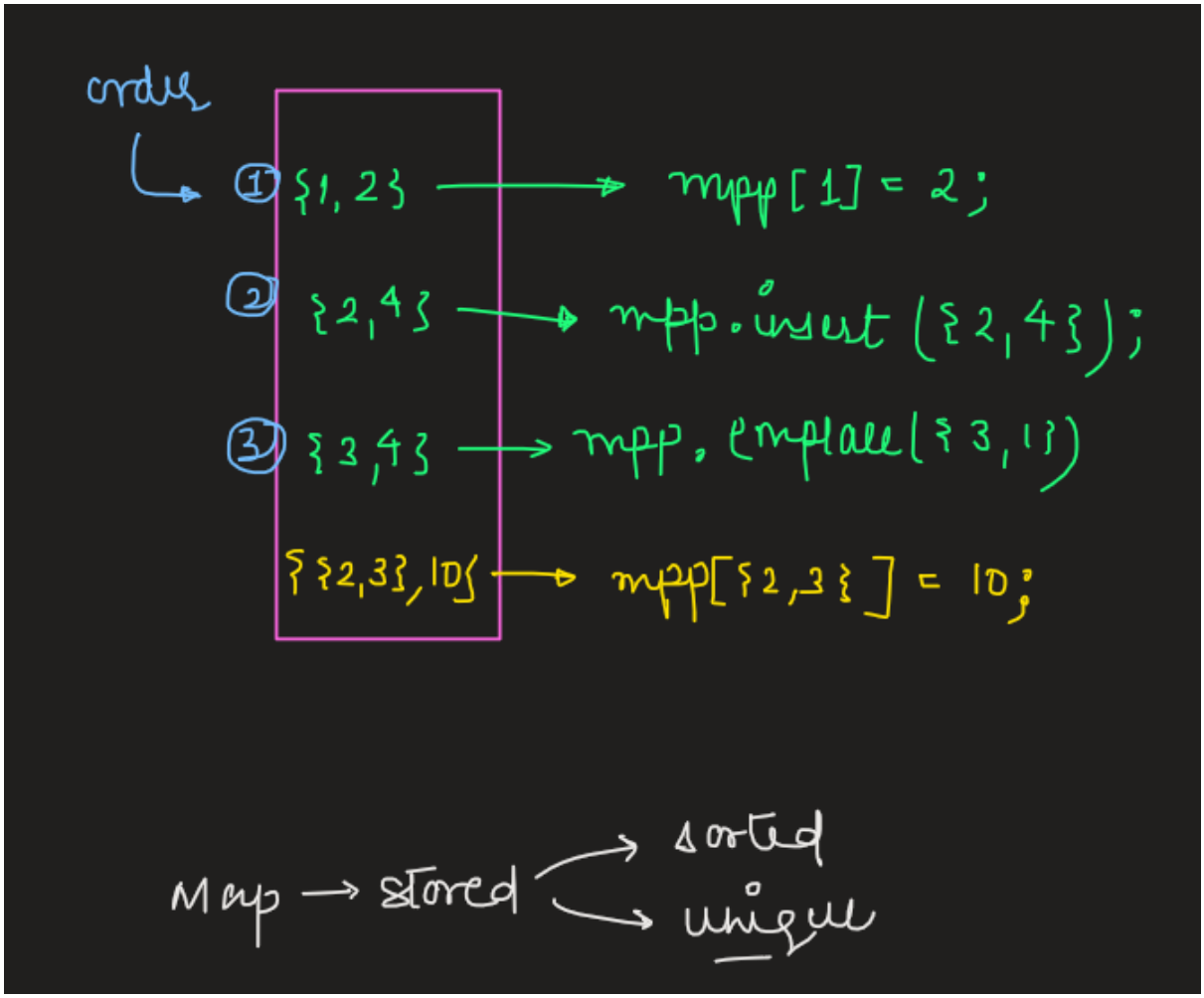
Syntax:

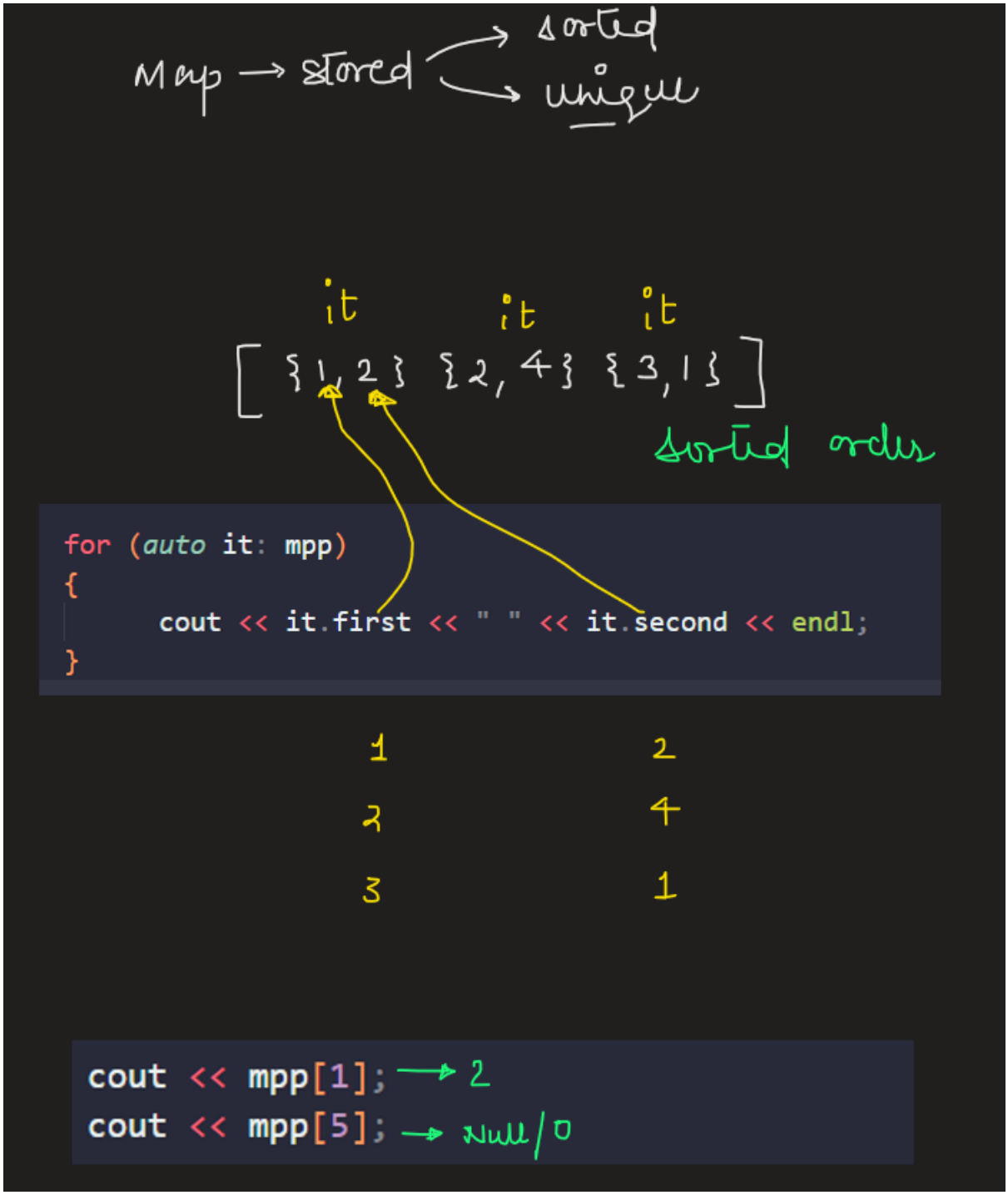
```
map<object_type,object_type> variable_name;
```

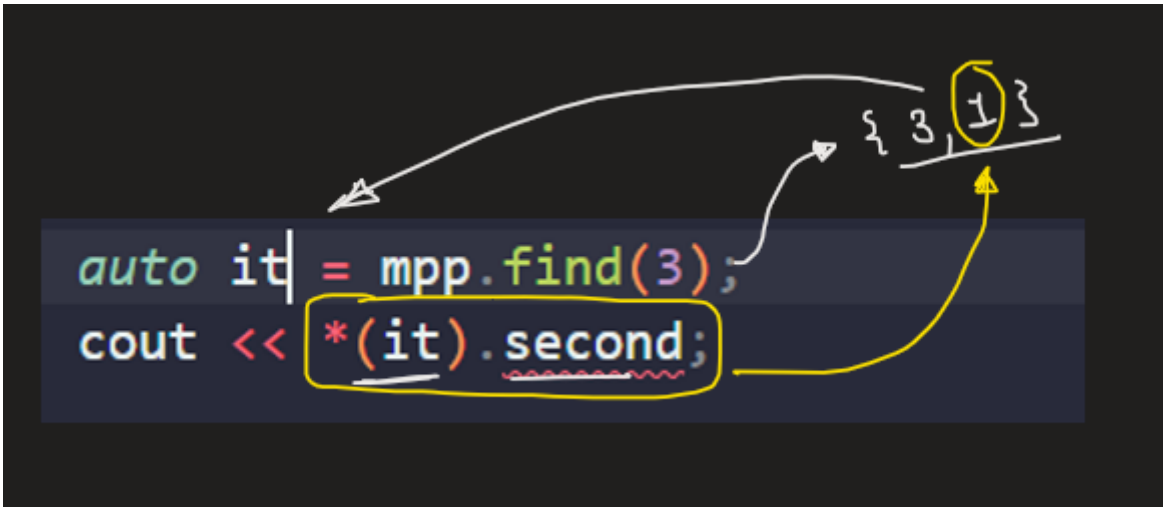
Example:

```
map<int,int> mpp;
map<string,int> mpp;
```


map < int, int > mpp;
map<int, pair<int, int>> mpp;
map<pair<int, int>, int> mpp;
Key
Value


order
1 {1, 2} → mpp[1] = 2;
2 {2,4} → mpp.insert({2,4});
3 {3,4} → mpp.emplace(3,1)
{{2,3},10} → mpp[{2,3}] = 10;
Map → stored → sorted
→ unique


Map → stored → sorted
→ unique
it
it
it
{1, 2} {2, 4} {3, 1}
sorted order
for (auto it: mpp)
{
cout << it.first << " " << it.second << endl;
}
1 2
2 4
3 1
cout << mpp[1]; → 2
cout << mpp[5]; → null/0



The Whole Code —

```
// Containers--> Maps
#include <bits/stdc++.h>
using namespace std;

// Map --> respect of key and value {key, value} { key --> ds, int, double, pair}
void explainMap()
{
      map < int, int > mpp;

      map<int, pair<int, int>> mpp;

      map<pair<int, int>, int> mpp;

      mpp[1] = 2; //  Store 2 on key 1 --> {{1,2}}

      // {
      //       {1,2}
      //       {2,4}
      //       {3,1}
      // }

      for (auto it: mpp)
      {
            cout << it.first << " " << it.second << endl;
      }

      cout << mpp[1];
      cout << mpp[5];

      // auto it = mpp.find(3);
      // cout << *(it).second;

      auto it = mpp.find(5); // return mpp.end()

      // This is the syntax
```

```
	auto it = mpp.lower_bound(2);
	auto it = mpp.upper_bound(3);

	// erase, swap, size, empty, are same as above

}
int main()
{
	explainMap();
	return 0;
}
```

Functions in map:

insert() – to insert an element in the map.

```
map<int,int> mp;
mp.insert({1,10});
mp.insert({2,20});
```

begin() – return an iterator pointing to the first element in the map.

```
mp.begin();
```

end() – returns an iterator to the theoretical element after the last element.

```
mp.end();
```

clear() – deletes all the elements in the map.

```
mp.clear();
```

find() – to search for an element in the map.

```
map<int,int> mp;
mp.insert({1,10});
mp.insert({2,20});
if(mp.find(2)!=mp.end())
cout<<"true"<<endl;
```

erase() – to delete a single element or elements between a particular range.

```
mp.erase(key);
mp.erase(iterator position);
mp.erase(iterator position 1, iterator position 2);
```

size() – returns the number of elements on the map.

```
mp.size();
```

empty() – to check if the map is empty or not.

```
mp.empty();
```

Striver’s Code:

```
#include<bits/stdc++.h>

using namespace std;

int main() {
  map < int, int > mp;
  for (int i = 1; i <= 5; i++) {
    mp.insert({i , i * 10});
  }

  cout << "Elements present in the map: " << endl;
  cout << "Key\\tElement" << endl;
  for (auto it = mp.begin(); it != mp.end(); it++) {
```

```
    cout << it -> first << "\\t" << it -> second << endl;
  }

  int n = 2;
  if (mp.find(2) != mp.end())
    cout << n << " is present in map" << endl;

  mp.erase(mp.begin());
  cout << "Elements after deleting the first element: " << endl;
  cout << "Key\\tElement" << endl;
  for (auto it = mp.begin(); it != mp.end(); it++) {
    cout << it -> first << "\\t" << it -> second << endl;
  }

  cout << "The size of the map is: " << mp.size() << endl;

  if (mp.empty() == false)
    cout << "The map is not empty " << endl;
  else
    cout << "The map is empty" << endl;
  mp.clear();
  cout << "Size of the map after clearing all the elements: " << mp.size();
}

Output:

Elements present in the map:
Key Element
1 10
2 20
3 30
4 40
5 50
2 is present in map
Elements after deleting the first element:
Key Element
2 20
3 30
4 40
5 50
The size of the map is: 4
The map is not empty
Size of the map after clearing all the elements: 0
```

Other functions:

- **cbegin()** – it refers to the first element of the map.
- **cend()** – it refers to the theoretical element after the last element of the map.
- **rbegin()** – it points to the last element of the map.
- **rend()** – it points to the theoretical element before the first element of the map.

- **emplace()** – to insert an element in the map.
- **max_size()** – the maximum elements a map can hold.