→ pair<int, int> p = {1, 3};

p.first → 1 p.second → 3

→ pair<int, pair<int, int>> p = {1, {3, 4}};

p.first → 1 p.second.first → 3 p.second.second → 4

→ pair<int, int> arr[] = {{1,2}, {2, 5}, {5, 1}};

arr[1].second = {2, 5}

→ vector<int> v; // creat empty container

v.push_back(1);
v.emplace_back(2); // it is faster then v.push_back

→ vector<pair<int, int>> v;

v.push_back({1, 2});
v.emplace_back(1, 2); // here we not need of {}

→ vector<int> v(5); // create 5 size array with garbadge value

→ vector<int> v(5, 2); // create 5 size array with 2 in every place

→ vector<int> v1(v); // create a copy of v

// iterate array

→ vector<int>:: iterator it = v.begin(); // to the first element of the array it give address

cout << *it; // it print the value

→ vector<int>::iterator it = v.end();
// it points to next of ending index of vector
to point it to end element we have --it;

→ vector<int>::iterator it = v.rend();
// it points to the ending index of vector

→ vector<int>::iterator it = v.rbegin();
// it points to the index before the starting index
of the array

→ v.at[0] , v[0] are same

→ v.back() // it points to the last index of vector

```
for( vector<int>::iterator it = v.begin(); it != v.end(); it++)
{
  cout<< *(it) << " "; }

for(auto it = v.begin(); it!=v.end(); it++)
{
   cout << *(it) <<" ";
}

// for each loop

for( auto it : v)
{
   cout<< it<<" ";
}
```

→ v.erase(v.begin()); // it delete first element

→ v.erase (v.begin()+1, v.begin()+4);
// it delete elements [2, 3] it cant delete 4th

→ v.insert(v.begin()+1, 200);
// it insert 200 at 1st index

→ v.insert(v.begin()+1, 2, 10);
// it insert 10 at 1st and 2nd index (2 times)

→ v.insert(v.begin(), copy.begin(), copy.end());
position where to insert | Starting of copy vector | ending of copy vector

// it insert copy vector in v vector in specific location that we give here

→ v1.swap(v2);
// it swaps the both vectors

→ v.size();
→ v.clear();
→ v.empty();
→ v.pop_back()

list // this is doubly linked list

→ list<int> l;

every node address type

→ l.push_back(2);
→ l.emplace_back(4);

list<int>::iterator

→ l.push_front(5);
→ l.emplace_front(5);

→ l.emplace_front(); {2,4};

All all vector function's are also applicable here

// it is dabley linked list

Deque

→ deque<int> dq;

it similary to list and vector all function are applied here

Stack

→ stack<int> s;

→ s.push(I);

→ s.emplace(5);

→ s.pop();

→ s.top();

→ s.size()

→ s.empty()

→ stack<int> s1, s2;
// make 2 empty stacks

→ s1.swap(s2);
// it swap to stacks

Queue

→ queue <int> q;

→ q.push(1);

→ q.emplace(2);

→ q.pop();

→ q.back();
// last elementof queue

→ q.front();
// front elent of queue

priority queue

→ priority_queue <int> pq;

→ pq.push(5); // 5
→ pq.push(8); // 85
→ pq.push(2); // 852

→ pq.emplace(10);

→ pq.top();
// it gives the topest element

→ pq.pop(); // topest element is poped

→ priority_queue <int, vector<int>, greater<int>> pq ;

// to make a queue that's top element always a smallest element

Set

→ set<int> s ;

→ s.insert(1);

→ s.insert(1); // it don't store duplicate elements

→ auto it = s.find(3); // it points of address of 3

// if element not found it return s.end();

→ s.erase(5); OR s.erase(it);

→ s.erase(s.find(2), s.find(4));

→ s.count(1); // if it found it give 1 othwise give 0

→ auto it = s.lower_bound(2);

→ auto it = s.upper_bound(3);

// And all vector function's are also applied here

Multiset

→ multiset <int> ms ;

→ ms.insert(1) ; // 1
→ ms.insert(1) ; // 1 1

→ ms.erase(1) ; // it erase all 1's

→ ms.erase(ms.find(1)) ; // it erase first 1

→ ms.erase(ms.find(2), ms.find(2)+2) ;

// all set function are applied here

unordered set

→ unordered_set<int> st ;

// all multiset operations are applied here
but lower_bound() and upper_bound()
not work

Map

→ map<int, int> m ;

→ map<int, pair<int, int>> m ;

→ map<pair<int, int>, int> m ;

→ m[1] = 2 ; // 1 → 2

→ m.emplace({3, 1}) ; // {3, 1}

→ m.insert({2,4}); // {2,4}

iterate

```
for( auto it: m )
{
    cout << it.first << " " << it.second << " ";
}
```

→ m[1]; // it give value of 1
// it it not found 1 than it give 0

→ auto it = m.find(3);

cout << *(it).second;

//if it not found 3 than it return m.end();

→ auto it = m.lower_bound(2);

→ auto it = m.upper_bound(3);
it find next highest key to 3 if it not found 3
it give m.begin();
// map store unique key in sorted order

// all other functions are same
erase, swap, size, empty

→ m[outerkey][innerkey] = value;
if we have map that also contain map

Multimap

```
multi_mp<int, int> m
```

// duplicate key in map

Unordered map

```
unordered_map<int, int> m;
```

// unique key but not sorted

Algorithms

→ `sort(v.begin(), v.end());`

→ `sort(it+2, it+5);`

→ `sort(v.begin(), v.end(), greater<int>());`
// sort in decreasing order

→ `sort(v.begin(), v.end(), cmp);`
↓
boolean function

```
static bool(pair<int, int> p1, pair<int, int> p2)
{
    if(p1.second < p2.second)
      return true;
    if(p1.second > p2.second)
      return false;
    if(p1.second == p2.second)
    {
      if(p1.first > p2.first)
```

```
        return true;
        return false;
    }
}
```

→ // Count how many 1's in the binary no

```
int num = 7;
int cnt = __builtin_popcount();
```

→ // if no in long long

```
long long num = 165786578687;
int cnt = __builtin_popcountll();
```

→ // find permutation

```
string s = "123";
// always start from sorted no
do{
    cout<<s<<endl;
} while(next_permutation(s.begin(), s.end()));
```

→ // max element

```
int maxi = *max_element(v.begin(), v.end());
```

→ // min element

```
int mini = *min_element(v.begin(), v.end());
```

→ // search element in is array or not

```
bool ans = binary_search(v.begin(), v.end(), 3);
```

→ // lower bound

```
v = [1, 4, 5, 6, 9, 9];

int index = lower_bound(v.begin(), v.end(), 4) - v.begin();
```

// it give the index of first 4 present in vector

```
int index = lower_bound(v.begin(), v.end(), 7) - v.begin();
```

// when it not found 7 than it find next immidate greater element than 7 i.e 9 so it return 4

```
int index = lower_bound(v.begin(), v.end(), 10) - v.begin();
```

// when it not found 10 than it find next immidate greater element than 10 that is also not found so it return index 6 i.e is out of vector

// upper_bound

```
int index = upper_bound(v.begin(), v.end(), 4) - v.begin();
```

// it return next greater element than 4 i.e 2 index

```
int index = upper_bound(v.begin(), v.end(), 7) - v.begin();
```

// it element not found it return next greater element than 7 i.e 4 index

```
int index = upper_bound (v.begin(), v.end(), 0) - v.begin();
```

// it element not found it return next greater element if no greater element found it return index 6 i.e out of vector