Pairs

pair<data-type, data-type> p={1, 3}

pair<data-type, pair<data-type, data-type>>

p= {1, {3, 2}}

pair <data-type, pair data-type> arr[]=

{{1, 2}, {3, 4}};

p.first , p.second

p.first.second

Vector

dynamic memory.

unlike array memory can be increased

vector<data-type> v; → creates an empty contai

v.push_back(1); → {1}

v.emplace_back(2); → {1, 2}

↳ faster than push back

vector <pair <int, int>> vec;

v.push_back({1,2});

v.emplace_back(1,2);

vector<int> v(5, 100); → {100, 100, 100, 100, 100}

↓

vector already have 5 places filled with no. 100

Date: / /

vector <int> v(5); → a container of size 5 is created

vector <int> v2(v); → copy of v

after 5 we can increase the size of vector

cout << v[1] << v[0];
~~cout~~

vector<int>::iterator it = v.begin();
it++;
cout << *(it) << " ";
(it) ↳ gives the value in memory
v.begin() → pointing to the memory holding this

it = it + 2;
cout << *(it) << " ";

vector <int> :: iterator it = v.end();
v.end() → points somewhere after the last element

it--; → fir ye 15 ko point karega
{0, 20, 15} v.end
↑

vector <int> :: iterator it = v.rend();
v.rend() → reverse end

v.rbegin();

rend {0, 20, 15};
↑

{0, 20, 15}
↑
it++ rbegin

Date __/__/____

cout << v[0] << v.at[0]; → same meaning
cout << v.back() ~~<<~~;

↓
{10, 20, 30}
↳ prints the last element

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

for (auto it = v.begin(); it != v.end; it++);

↓
according to data, it auto assing the value to iterara

for (auto it : v){ → for each loop
    cout << it << " ";

erase

v.~~erase~~ (v.begin()+1);
v.erase (v.begin()+1, v.begin()+4); → begin+4
↳ given (10, 20, 30, 40, 50, 60)
↳ after erase (10, 50, 60)

vector<int> v(2, 100); → (100, 100)
~~vector~~
~~v.inse~~ v.insert (v.begin(), 300); → (300, 100, 100)
v.insert (v.begin(), 2, 10); → (10, 10, 300, 100, 100)

v.size();
v.pop_back(); → last element popped
v1.swap(v2); $v_1 \to (10, 20)$ ~~..~~ = after swap
$v_2 \to (30, 40)$ $v_1 \to (30, 40)$ v_2 (

Date. _/_/_

v.clear(); → clears empty
v.empty(); → answers question like is vector empty or not

list

```
list<int> ls;
ls.push_back(2);
ls.emplace_back(4);
ls.push_front(5);
ls.emplace_front(); {2, 4};
```

dequeue

```
deque <int> dq;
~~deque~~ dq.emplace_back();
dq.push_back
dq.push_front();
dq.emplace_front();
dq.pop_back();
dq.pop_front();
```

// rest same as vector functn swap and all

stack → LIFO

```
stack <int> st;
st.push(1); → {1}
st.push(2); → {2, 1}
st.push(3); → {3, 2, 1}
```

Date __/__/____

st.pop(); → {2, 1}

st.top(); → 2   st[2] not allowed
st.size();
st.empty();

stack<int> st1, st2;
st1.swap(st2);

Queue → FIFO

queue<int> q;
q.push(1); → {1}

q.push(2); → {1, 2}
q.push(3); → {1, 2, 3}

q.back() += 5; → {1, 2, 8}
q.front(); → 1
q.back(); → 8

q.pop(); → {2, 8}

push or emplace same same in stack & queue, pq

Priority queue → lexicographically or in descending order

priority_queue<int> pq;
pq.push(5) → {5}
pq.push(2) → {5, 2}
pq.push(18) → {18, 5, 2}
~~pq.pop()~~ pq.top(); → 18
pq.pop(); → {5, 2}

Date. _/_/_

//If we want in decending order

```
priority_queue<int, vector<int>, greater<int>> pq;
pq.push(5);   {5}
pq.push(2);   {2,5}
pq.push(18);  {2,5,18}
```

<u>Set:</u> → sorted in unique

```
set<int> st;
st.insert(2) → {2}
st.insert(3) → {2,3}
st.insert(2) → {2,3}
insert(1) → {1,2,3}

auto it = st.find(3);        → iterator to
                               find 3.
st.erase(5);
int cnt = st.count(2);       → 1 if exist
                             → 0 if does not exit

auto it1 = st.find(2);
auto it2 = st.find(3);
st.erase(it1, it2);

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

Date.__/__/____

lower_bound / upper_bound

{1 2 3 3 5 6 7 9}

lower_bound (3) → since it is in stack
lower_bound (4) → since it is not in stack but a number just bigger than it is
lower_bound (10)
since it is not in stack and a no just bigger than it is also not there

{1, 3, 3, 4, 5, 6, ~~7~~, 8, 9}

upper_bound (4);
upper_bound (7);
upper_bound (10);
} always put a no. just bigger irrespective if it's present or not

no. bigger not present

multiset → sorted not unique

multiset <int> ms;
ms.insert (1); → {1}
ms.insert (1); → {1, 1}
ms.insert (1); → {1, 1, 1}
ms.insert (2); → {1, 1, 1, 2}
int cnt = ms.count (1); → 3
ms.erase (ms.find(1)); → {1, 1, 2}
ms.erase (1); → {2}

Date: / /

unordered set : unique but not sorted

same as set

Map → stores unique keys in sorted order

Map : unique key → value → can be repeated

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

mpp[1]=2;        → or {1, 2}
mpp.emplace({3,1});
mpp.insert({2,4});

mpp[{2,3}]=10;

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

mpp
{1 , 2}
{2 , 4}
{3 , 1}

```
cout<< mpp[1];   → 2     (1 → key)
cout<< mpp[5];   → NULL

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

Date __/__/____

multimap → can store duplicate keys in sorted order

// only mpp[key] cannot be used here.

unordered_map() → randomized unique

Sorting

```
sort(a, a+n);              } ascending
sort(v.begin(), v.end());  }
sort(a, a+n, greater<int>()); → descending
```

Q. sort a[] = {{1,2}, {2,1}, {4,1}}; according to second element. If second element is same then according to first elements in descending

i.e

sorted : {{4,1}, {2,1}, {1,2}}

```
sort(a, a+n, comp);
bool comp(pair<int, int> p1, pair<int, int> p2){
    if(p1.second < p2.second) return true;
    if(p1.second > p2.second) return false;
    if(p1.first > p2.first) return true;
    return false;
}
```

Date. __/__/____

Q. find ~~a~~ all permutation
s = "4321";

① sort (s.begin(), s.end());
② do{
cout << s << endl;
}while(next_permutation(s.begin(), s.end()));

Q max element of an arr
* max_element (a, a+n);
* min_element (a, a+n);