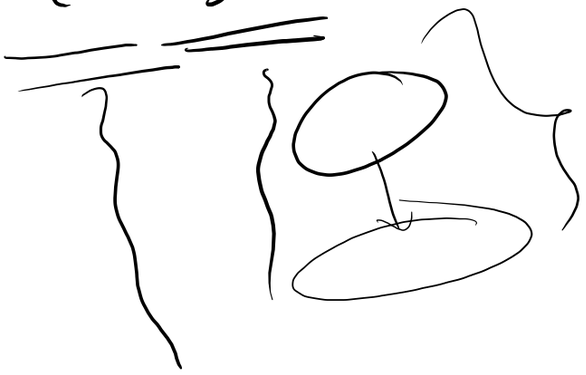
C++ STL



Standard template library

- containers
- Iterators
- Algorithms (in built)
- functors → (lambda functions)

Containers

Sequential → Just like array

→ vectors
→ stack
→ queue
→ pair (not a container) C++ class

Ordered containers:

→ maps
→ multimap
→ set
→ multiset

unique & Sorted

Unordered Containers:

→ unordered map
→ unordered set

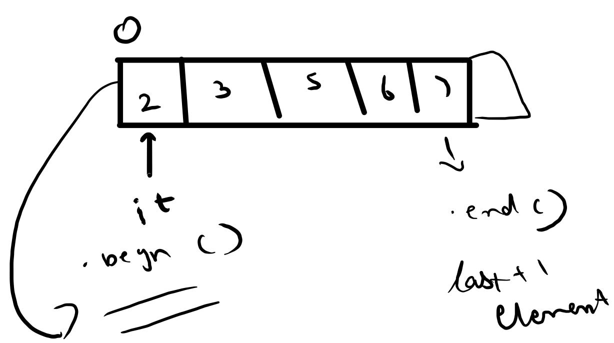
} hashing type

→ Iterators:
similar to Pointer
↓
variables like memory
like address ke point
~~at~~ karte the.

but in containers to Point
an element in we
don't use
Pointers.

vector<int> :: iterator it;

.begin(), .end()




```
int main() {
    vector<int> v = {2, 3, 5, 6, 7};
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i] << " "; }
    cout << endl;

    vector<int> :: iterator it = v.begin();
    cout << (*it) << endl;

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

C++ maps

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    map<int, string> m;
    m[1] = "abc";
    m[5] = "cde";
    m[3] = "acd";
    m[6] = "a";
    m[5] = "coe";
    print(m);
}
void print(map<int, string> &m) {
    cout << m.size() << endl;
    for (auto &pr : m) {
        cout << pr.first << " " << pr.second << endl;
    }
}
```

unique & sorted

m[5]

automatic assumed

Output

1	abc
3	acd
5	coe
6	a

Given: N strings, Print unique strings in lexicographical order with their frequency

$N < 10^5$

Input:

8
abc
def
ghi
abc
def
jBA
xyz
mno

Output

abc	2
def	2
jBA	1
jhc	1
mno	1
xyz	1

```
int main() {
    map<string, int> m;
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        string s;
        cin >> s;
        m[s]++;
    }
    for (auto pr : m) {
        cout << pr.first << " " << pr.second << endl;
    }
}
```

C++ inbuilt algorithms :

1) min_element :

```
int main () {
    int n;
    cin >> n;
    vector <int> v(n);
    for (int i=0 ; i<n ; ++i) {
        cin >> v[i];
    }
```

1) `int min = min_element (v.begin(), v.end());`
`cout << min << endl;`

2) `int max = max_element ( v.begin() , v.end())`
`cout << max << endl;`

3) `int sum = accumulate (v.begin (), v.end() , 0 );`
`cout << sum << endl;`

4) `int ct = count (v.begin(), v.end(), 7)`

1 2 3 4 5 5 6 6
2 (77)

5) `auto it = find (v.begin, v.end(), 10)` → 2
`if ( it != v.end () )`
`cout << it << endl;`
`else`
`cout << "Element not found" << endl;`

6) `reverse (v.begin() v.end ())`
`for (auto val : v)`
`cout << val << " ";`
`} cout << endl;`
`}`

`string s = "abcdefgh";`
`reverse ( s.begin (), s.end());`
`cout << s << endl;`

`reverse ( s.begin() + 1, s.end ());`

Lambda functions :

small syntax for writing temporary type function

`[] (int x) { return x*2; } (2);`

Lambda function functor

```
int main () {
cout << [] (int x) { return x+2; } (2);
}
```

return: output: 4

1) all_of

```
int main () {
vector <int> v = { 2, 4, 5};
cout << all_of (v.begin(), v.end());
```

input: 1 2 3 4 5 6

`[] (int x) { return x>0; });`

2) any_of

`cout << any_of (v.begin(), v.end())`

input: -1 -2 -3 -4

5

3) None_of :

`cout << none_of (v.begin(), v.end())`