STL

C++ STL

- Algo
- Containers

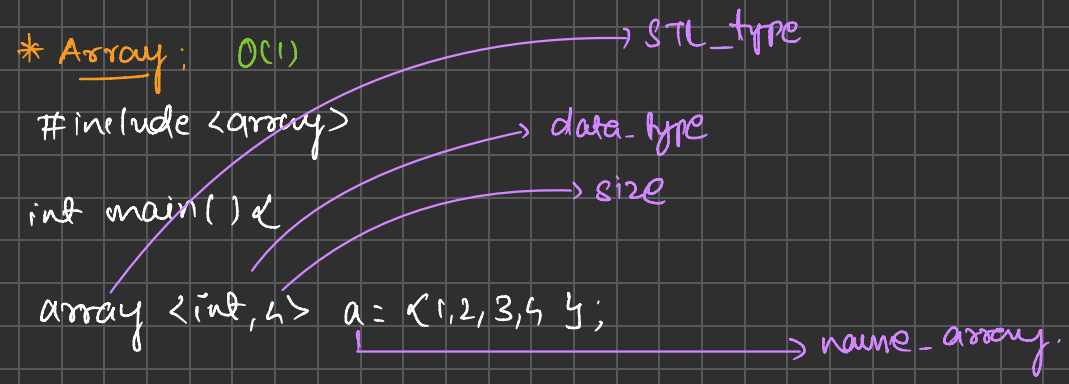
Containers

- Sequence Containers
  - Array
  - vector
  - deque
  - list
  - forward_list
- Container Adaptors
  - stack
  - Queue
  - Priority Queue
- Associative Containers
  - Set
  - Map
  - MultiSet
  - Multi Map
- Unordered Associative
  - Unordered set
  - Unordered Map
  - Unordered multiset
  - Unordered Multi Map

Algorithms

- binary search
- Lower / Upper bound
- min / max
- reverse / rotate
- sort / swap etc.

* Array ; O(1)



→ size() : a.size();

→ at() : a.at(2); → element at index 2

→ front() : a.front(); → first element

→ back() ; a.back(); → last element

* Vector ; → It is a dynamic array. O(1)

→ when size of vector is full and you insert another element then size of vector will be doubled.

→ size :- how many elements; → capacity: total allocated memory.

```
#include<vector>
int main(){
    vector<int> v;

}
```

→ capacity() ; v.capacity();

→ push_back() : v.push_back(5);

→ pop_back() : v.pop_back();

→ size() : v.size();

→ front() : v.front();

→ back() : v.back();

→ begin() : v.begin();

→ end() : v.end();

→ sort() : sort(v.begin(), v.end());

→ at();

→ vector <int> a(5, 1); → size
→ every element assign as 1

→ vector <int> last(a); → vector a copy in last.

* deque : O(1)

```
#include <deque>

int main(){
    deque <int> d;

}
```

→ push_back() :
→ push_front() :
→ pop_back() :
→ pop_front() :
→ empty() : d.empty() → true/false
→ begin() :
→ end() :
→ front() :
→ back() :
→ erase() : d.erase(d.begin(), d.end());
→ size() : d.size()

* list : doubly linked list. , can't use .at() O(1)

```
#include <list>

int main(){

    list<int> l;

    l.push_back(1);
    l.push_front(2);

    for(int i:l){
        cout<<i<<" ";
    }
```

→ push_back():
→ push_front():
→ pop_back():
→ pop_front():
→ l.erase(l.begin):
→ size():
→ front():
→ begin():
→ end():

* stack :  LIFO (Last in first out). O(1)

```
#include<stack>

int main(){

    stack<string> s;

        s.push("love");
        s.push("babbar");
        s.push("kumar");
```

};

→ push():
→ pop():
→ top():
→ size():
→ empty():

* Queue: → line, First in First Out (FIFO) O(1)

```
#include <queue>

int main(){
    queue <string> q;

};
```

→ push():
→ pop():
→ size():

* priority queue: O(1)
  - (default). max heap → first element will be greatest
  - min heap

```
#include <queue>

int main(){

    // max-heap
    priority_queue<int> maxi;

    // min-heap
    priority_queue<int, vector<int>, greater<int> > mini;

};
```

→ top():
→ pop():
→ push():
→ size():
→ empty():

* set(); → store Unique elements only. O(1)
  ↳ Implement by using BST.
  ↳ return element in sorted order.
  ↳ can't be modify.

```
#include<set>
int main(){
    set<int> s;
    s.insert(6);
    s.insert(2);
}
```

→ insert():
→ find():
→ erase():
→ count(): s.count(6); → present or not → 1/0

```
set<int>::iterator it = s.begin();
it++;
s.erase(it);
```

```
set<int> :: iterator itr = s.find(5);
```

*** Map :** - store in key value pair. O(log n) balance tree

- each key is unique.
- per key only one value.
  - → value can be same for 2 key.
  - → key can't be same for 1 value.

```
#include <map>

int main(){

    map<int, string> m;

    m[1] = "love";
    m[2] = "babbar";
    m[3] = "kumar";

    for(string i: m){
        cout << i.first << endl;
    }

    m.insert({5, "bheem"});

    m.count(3);

    m.erase(3);

    auto it = m.find(5);

    for(auto i=it; i != m.end(); i++){
        cout << (*i).first << endl;
    }

}
```

key

value

gives key.
i.second → value

* Algorithms:

```
#include <algorithm>
#include <vector>

int main() {
    vector<int> v;
    v.push_back(1);
    v.push_back(3);
    v.push_back(6);
    v.push_back(7);

    binary_search(v.begin(), v.end(), 6);
    lower_bound(v.begin(), v.end(), 6) - v.begin();
    upper_bound
    - - - - - - -

    max(1, 6);
    min(2, 3);
    swap(x, y);
    string s = "abcd";
    reverse(s.begin(), s.end());
    rotate(v.begin(), v.begin()+1, v.end());
    sort(v.begin(), v.end());
```

→ Intro Sort
Quick Sort + heap, Insertion Sort