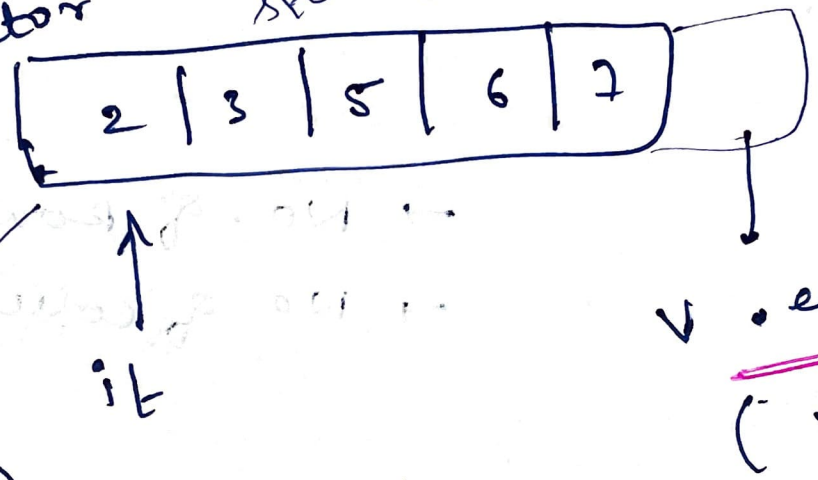
Iterators → used to accesses value of containers

vector → data is store in continuous block



declaration vector<int> :: iterator it = v.begin();

```
cout << (*it) << endl;
cout << (*(it+1)) << endl;
```

o/p: 2 3 5 6 7

2

3

To Print elements of vector using iterator :

```
vector < int > :: iterator it = v.begin();
```

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

by using iterators

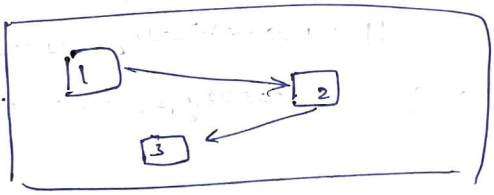
Here

it ++ → moves to next iterator

it +1 → next location.

Map

data stored is Not continuous



⇒ when iterator pointing to a pair

```
vector < pair < int, int>> v_p = {{1,2}, {3,4}, {2,3}}
```

```
vector < Pair < int, int>> :: iterator it ;
```

```
For ( it = v_p.begin() ; it != v_p.end(); ++it)
{ cout << ( it → First) << " " << (it → second) << endl;
}
```

o/p
12
34
23

(* it).First ⇔ it → First

Iterator code inshort (C++11 version)

Auto and Range based loops :

```
vector <int> v = { 2, 3, 5, 6, 7 } ;

for (int value : v)
{
    cout << value << " " ;
}
cout << endl;
```

Value is the copy of V vector

O/P : 2, 3, 5, 6, 7

& value is the actual values of v vector.

```
auto a = 1.0;
cout << a << endl;
```

auto // automatically determine a's datatype and int

Short form of iterator declaration → For pair & maps

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

declaration of iterators

Vector pair accessing through iterator

```
vector <pair <int, int> > v_p = { {1,2}, {2,3}, {3,4} }
for ( auto & value : v_p ) {
    cout << value.first << " " << value.second ;
}
```

→ duplication of keys are allowed

Maps, unordered maps, Multimaps :

→ Order matters

Map → Store key : value

int		string
1	→	abc
5	→	cde
3	→	acd

In maps sorted order of key is stored.

↓

Implemented by Red-Black Trees (comparision)

→ every element in a map is a pair, which stores key and value, and duplicate keys in the map can't be inserted ~~in map~~.

→ order doesn't matter

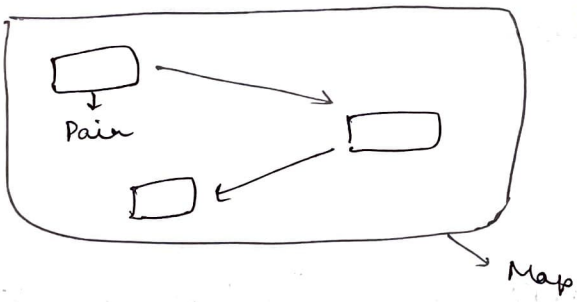
unordered map

↓

Here map's key is stored in unsorted order.

↓

Implemented by using hash tables (Pair & set can't be used)




```
int main () {

    map < int, string > m;

    m [1] = "abc";
    m [5] = "cde";
    m [3] = "acd";

}
```

To access value

(or) m.insert ({4, "afg"})

→ $O(\log(n))$

To print maps by long form of iterator

```
map < int, string > :: iterator it ;
for ( it = m.begin ; it != m.end () ; ++it)
{
    cout << (*it).first << " " << (*it.second) ;
}
```

o/p:

1 abc.
3 acd
4 afg
5 cdc

The Maps are printed in sorted order.

Shortcut to Print maps by iterator

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

we have m.size () → To print size of map

```
auto it = m.find (3) ;   //O(log(n))
    If ( it == m.end ()) {
        cout << "No value" ; }
else{ cout << (*it).first << " " << (*it.second) ; }
```

o/p: 3 acd.

Operations on map

```
m.size();
m.erase();
m.find();
m.clear();
```

Unordered Maps

declaration: unordered_map<int, string> m;

```
m[1] = "abc";   → // O(1)
m[5] = "cdc";
```

o/p: Print in random order 5 cdc
1 abc

Sets → uses red black trees → Key // set store the unique elements in sorted order

Maps → pair → Key & value

```
int main () {
    set<string> s;          // O(log(n)) → ordered set
    s.insert("abc");
    s.insert("bcd");        unordered set → O(1)
    s.insert("zsdf");       Multiset → O(log(n))
    auto it = s.find("abc");
}
```

allow duplicate keys to print

eg: abc abc def

(Priority queue)

→ Nesting in STL Maps and sets

∵ Pair is used for Key Comparision.

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

Key value

```
Pair < int, int > P1 , P2 ;
P1 = { 2, 2} ;
P2 = { 2, 3} ;
cout << ( P1 > P2 ) ;
}
```

O/P : 0 (False)

```
map < set < int >, int > m ;
```

Key value

```
Set < int > S1 = { 1, 2, 3 } ;
Set < int > S2 = { 2, 3 } ;
cout << ( S1 < S2 ) ;
```

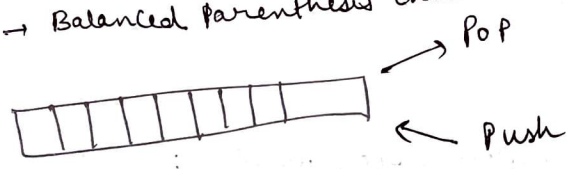
O/P : 1 (True)

Eg: map < pair <string, string>, vector <int> > m;

O/P: James Brown 25 35 40 (marks of 3 subjects)

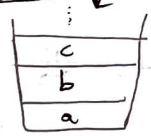
20/Feb

Stack → Balanced Parenthesis check, finding next greater element.



→ LIFO

Eg:

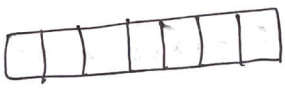


① Push
② Pop (Top)
③ Top

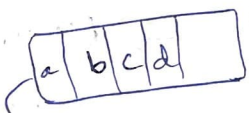
→ size
→ top element

queue

enqueue →



→ Dequeue → FIFO



① Push
② Pop (Front)
③ front

STL inbuilt implementation:

```
int main() {
        stack <int> s;
        s.push(2);
        s.push(3);
        s.push(4);
        s.push(5);
        while (!s.empty()) {
                cout << s.top();   || To print first element by popping the stack.
                s.pop();
        }
```

O/P: 5
4
3
2

Yes not empty (return bool true) if empty

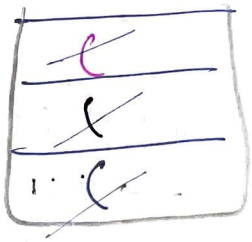
O/P: abc
bcd
cde
def

```
int main() {
        queue <string> q;
        q.push("abc");
        q.push("bcd");
        q.push("cde");
        q.push("def");

        while(!q.empty())
        {
                cout << q.front() << endl;
                q.pop();
        }
```

① Balanced Bracket Matching

(((()))



))))

→ If stack is empty after whole traversal then we can say Balanced bracket matching

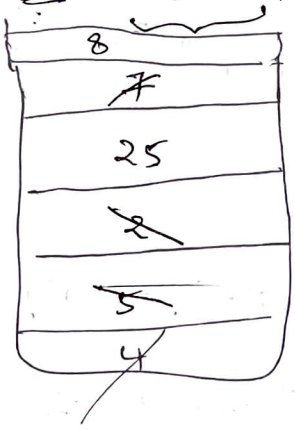
→ All opening brackets are added into the stack, if any closed bracket corresponding to it is present then that open bracket will get popped.

② Next greater Element using stack

NGE

4, 5, 2, 25, 7, 8

NGE 5, 25, 25, -1, 8, -1



→ **Introsort** → ① Take array I/p

② Sort using inbuilt sort algorithm.

Eg: sort (a, a+n); sort (start pointer of sorting , end + 1 pointer of sorting)

→ Sort in increasing order

Introsort = mixture of 3 sorting algorithms

= best sorting Algo.

quick sort + heap sort + insertion sort = Introsort

① Take vector of I/p

② sort (a.begin() , a.end()); → (n log n)

comparator function

sort (a.begin(), a.end(), should_i_swap);

Custom behaviour for sorting is achieved by using comparator function

To swap → return false

To not swap → return true

```
for (int i=0 ; i<n ; ++i) {
    for (int j = i+1 , j<n ; ++j) {
        if ((a[i] > a[j])) {
            swap(a[i], a[j]);
        } } }
```

// To print in ascending order

Should_i_swap(a[i], a[j])

comparator function.

o/p: 2 4 5 7 8 25

Function declared before main();

```
bool Should_i_swap (int a, int b) {
    if (a > b) return true;
    return false; }
```

(or) return a > b;

(or) return a < b;

Upper bound and Lower Bound in C++ STL

↳ Array (or) vector need to be sorted before

↳ log(n)

Sorted Array : 4 5 5 7 8 25

lower bound (7) → Find 7 if it is present (or) if 7 is absent it finds 8.

upperbound (7) → Every though 7 is present it will finds 8 only.

↳ lower bound and upper bound functions return location of element, in arrays → return pointer (int *ptr = ...)

in vectors → return iterator (auto it..)

For array

① sort (a, a+n)

starting address ↑ next of end address

② int *ptr = lower_bound (a, a+n, 7) → O(log(n))

cout << (*ptr) << endl; // o/p : 7

③ int *ptr = upper_bound (a, a+n, 7) → O(log(n))

cout << (*ptr) << endl; // o/p : 8

④ for using maps and sets for upperbound & lower Bound

auto it = s.lower_bound ( )

set <int> s;

Inbuilt Algorithm : → O(n) for array of vectors.

1. min element
2. Max element
3. Sum of array
4. Count of element.

↳ Take vector I/P

↳ min_element (v.begin(), v.end());

↳ returns address (iterator)

↳ Thus, write it as

(1) int min = *min_element (v.begin(), v.end());

I/P: 2 3 1 6 7 6

O/P : 1

(2) int max = *max_element (v.begin(), v.end());

O/P: 7.

initial sum

(3) int sum = accumulate (v.begin(), v.end(), 0);

O/P: 25 (2+3+1+6+7+6+0)

(3) int count = count (v.begin(), v.end(), 3);

O/P: 1 (No. of 3's in the vector).

(4) int element = *find (v.begin(), v.end(), 2);

O/P: 2

(5) reverse (v.begin(), v.end());

for (auto val : v) {

cout << val << " ";

}

O/P: 6 7 6 1 3 2

(6) String s = "abcdsfdsjf";

reverse (s.begin(), s.end());

cout << s << endl;

o/p: fjsdfsdcba

→ Lambda function → temporary function.

1. all_of → all need to be Positive (return true)
2. none_of
3. any_of → any element can be positive (return true)

→ return true or false.

Function for writing sum of two numbers:

auto sum = [ ] (int x, int y) { return x+y; }

sum(4, 7);

o/p: 11.

(7) int main() {

vector<int> v = {2, 3, 5};

cout << all_of (v.begin(), v.end(), [ ](int x) {return x > 0;});

Taking every element in vector and applying the function

function

o/p: 1 (True)