STL

→ Container
→ Iterators
→ Algorithm
→ functors.

(1) Container

Sequential
↓ ↓ ↓ ↓
vector Stack Queue Pair

Ordered
↓ ↓ ↓
maps Multi map multiset

Unordered
↓ ↓
Unordered Unordered
map set

* Nested Container

vector < vector < ins >>

→ map < int, vector< int >>

→ set < pair < int, string >>

(2) Iterators

It is used to point element of container.

→ begin(), end()

→ vector < int > :: iterator it;

→ Continuity

(3) Algorithms

→ Upper bound, lower bound
→ Sort (comparator)
→ max-element, min-element
→ Accumulate
→ reverse
→ Count
→ find
→ next / prev permutation

Date
Page

⑨ Functors

→ Classes which can act as function.

* Pair and vectors

→ Pair is a class which stores two value.

```
main()
{
    pair<int, string> p;
    p = make_pair(2, "abc");
    cout<<p.first<<" "<<p.second;
// p = {2, "abc"};
int pair<int, string> &p1 = p;
p1.first = 3;
cout<<p.first;
int a[] = {1, 2, 7}
```

```
int b[] = {2, 3, 4}

pair<int, int> p_array[3];

p_array[0] = {1, 2}

p_array[1] = {2, 3}

p_array[2] = {3, 4}

for (int i = 0; i < 3; i++)
{
    cout << p_arr[i].first << " " <<
            p_array[i].second ;
}
```

→ Pairs can be assigned, copied & compared

Vectors

→ Array of blocks. and size of vector is dynamic in nature.

```
main()
{
    vector<int> v;
    int n;
    cin>>n;
    for(int i=0; i<n; i++)
    {
        int x;
        cin>>x;
        v.push_back(x);
    }
    printvec(v);
}
void printvec(vector<int>v)
{
    for(int i=0; i<v.size(); i++)
    {
        cout<<v[i];
    }
}
```

// vector's size can change

```
main()
{
    vector<int> v(10);
    cout << printvec << 0.00... 0
    v.push_back(3) 3 0000... }
```

Here reference is more important like 94 9 don't use & sign then 9 will be no copied

Nesting in vector

```
main()
{ vector<pair<int,int>> v
    = {{1,2},{2,3},{3,4}};
  printvec(v);
}
```

```
void printvec(vector<pair<int,int>>
                                &v)
{
    cout << v.size();
    for(int i=0; i<v.size(); i++)
    {
        cout << v[i].first << " " <<
                    v[i].second;
    }
}
```

do get input by user

```
vector<pair<int,int>> v;
int n;
cin>>n;
for(int i=0; i<v.size(); i++)
{
    int x,y;
    cin x,y;
    v.push_back({x,y});
    or
    v.push_back(make_pair(x,y));
}
```

Vector of Arrays ⇒

```
main()
{   int N; cin >> N;
    vector< int > V[N];
```

N vectors.

V1 — — —

V2 — —

V3 — — — —

```
for(int i=0; i<N; i++)
{   int x;
    cin x;
    for(int j=0; j<x; j++)
    {   int n;
        cin n;
        V[i].push_back(n);
    }
}
```

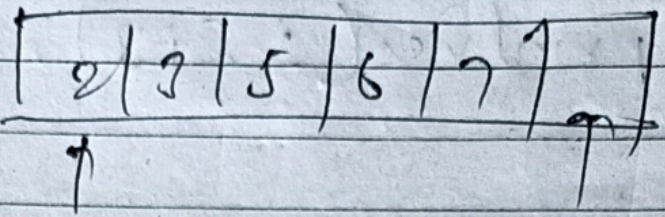
```
for(int i=0; i<n; i++)
{
    printvec(v[i]);
}

}
void printvec(vector<int> &v)
{
    cout<<v.size();
    for(int i=0; i<v.size(); i++)
    {
        cout<<v[i]<<" "<<endl;
    }
}
```

Here we can change row's elemat by ranging n.

Vector of vector of int

It is used to make changes size of each rows.

(71) Vector



it

begin()

v.begin()

↓

v[0]

.end()

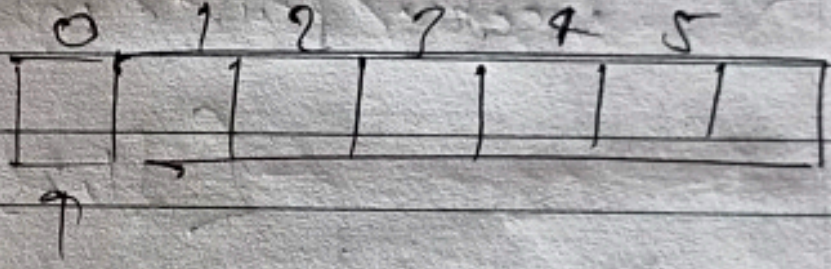
↓

next to last

```
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+1))<<endl;
for(it=v.gin(); it!=v.end();
                        ++it)
{   cout<<(*it)<<endl; }
```

int vector



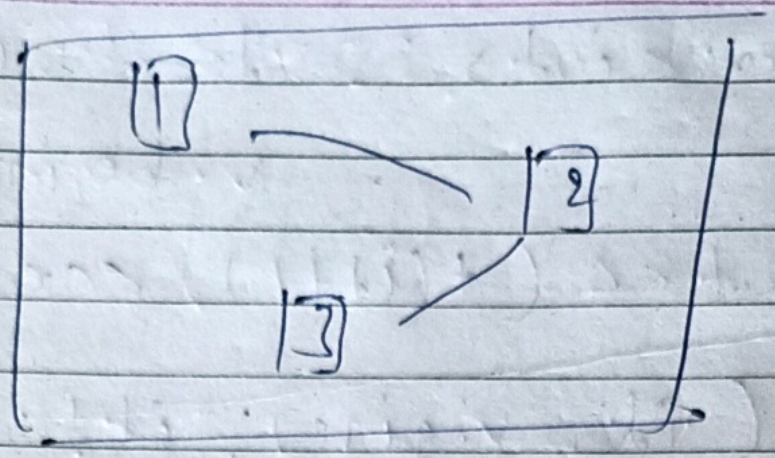
v.begin()

it++ => next iterator

it+1 → next location.

In case of map/set

it+1 is invalid oper




```
int
main()
{
    vector<pair<int, int>> v_p =
        { {1,2}, {2,3}, {3,4} };
    vector<pair<int, int>>::iterator
        it;
    for (it = v_p.begin(); it != v_p.end();
        ++it)
    {
        cout << (*it).first << " "
            (*it).second;
    }
```

or

```
cout<<(it->first)<<" "<<
      (it->second);
```

This is used when it is used in pair form.

i.e (*it).first = (it->first)

Auto and range based loops

(1) range based loop

```
vector<int> v={1,2,3,4,5}
for(int value : v)
{   cout<<value<<" "; }
cout<<endl;
```

we are just taking or copying 1, 2, 3, ... to values.

So, if we value++ no any work.

to change value

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

In case of pair

```
vector<pair<int, int>> v_p :
        {{1,2},{2,7}};
for( pair<int,int> &value : v_p)
{   cout << value.first << " "
        value.second; }
```

② Auto based loop

let auto a = 1;

Cout << a << 1

Code :-

```
vector<int> v = {1, 2, 3};
vector<int>::iterator it;
for(it = v.begin(); it != v.end(); it++)
{
    cout << *it << " ";
}
```

or

```
for(auto it = v.begin() ; )
{
    cout << *it;
}
```

In case of pair

```
vector<pair<int, int>> vp =
    {{1, 2}, {8, 3}, {3, 4}};
vector<pair<int, int>>::iterator it;
```

for (pair<int, int> it = v.begin();
— —)

```
{
    cout << (*it).first << " " <<
                (*it).second ;
}
```

or

```
for (auto it = v.begin — — )
{
    cout << it->first << " " << it->second;
}
```

(#) C++ STL MAPS

(1) maps (2) Unordered maps

→ maps is a data structure

(key , data)

int	string
1	abc
5	cdc
3	acd

In normal map, it will mapped with sorted order of keys. and this doesn't follows in unordered_map.

→ Element of a map is pair.

→ It is not continuous ds.

```
main()
{
    map<int, string> m;
    m[1] = "abc";
    m[5] = "cdc";
    m[3] = "acd";
        or
    m.insert({4, "afg"});
```

Date
Page

to Declare iterator.

map<int, string>::iterator it;

for(it = m.begin(); it != m.end(); it++)

{ cout<<(*it).first<<" "<<

(*it).second;

or

cout<<it->first<<" "<<it->second

;

}

→ We can't implement duplicate key.

→ Insertion and access time is O(logn)

auto it = m.find(3); ⇒ return it

Date
Page

```
if ( it == m.end() )
{
    cout << "no value";
}
else
{   cout << it->first << " " <<
        it->second;
```

```
m.erase();
```

key value it

```
int m.erase(7);
print(m);
```

1 abc
5 cde

```
auto it = m.find(7);
m.erase(it);
print(m);
```

if (it != m.end())
{ m.erase(it); }

~~m.erase~~
m.clear(); ⇒ it clear map.

Use of auto

for(auto it = v.begin() —)
{ cout << it->first << (*it).second }
or
for(auto it : v)
{ cout << it.first << it.second }

Use of Range

for(int value : v)
{ cout << value }.
in pair

value.

cout << ~~it~~ first;

Unordered maps & Multimaps

Insertion = O(1)

find(), erase() = O(1)

- There is no order.
- unordered_map<int, string> m;

We use unordered_map because it saves time and if doesn't matter with ordered.

Set, Ordered set & Multiset

Sets are set of keys. In maps we had keys as well as value, but in set we will have only keys

Cf

```
main()
{
  set<string> s;
  s.insert("abc");   // log(n)
  s.insert("zsdf");
  s.insert("bcd");
  auto it = s.find("abc");
  if(it != s.end())
  {
    cout<<(*it);
  }
  print(s);
}
void print(set<string> s)
{
  for(string val : s)
  {
    cout<<(*val);
  }
}
```

erase() func could take both iterator as well as value.

Now : unordered_set. O(1)

```
main()
{   unordered_set<string> s;
    s.insert("abc");
    s.insert("def");
    auto it = s.find("abc");
    if(it != s.end())
    {   cout<<(*it);}
    print(s);
}
void print(unordered_set<
            string> s)
{   for(auto it:s)
    {   cout<<(*it);}
}
```

Inside unordered set <p>

we cannot take set, pair in the except that we could put int, string.

⇒ Multiset O(log n)

```
multiset<string> s;
m.insert("abc");
m.insert("abc");
m.insert("def");
print(s);
```

→ Here duplicate printing is allowed.

→ It also print in sorted ordered.

→ If there are duplicate present then it just print iterator of first part. This part is done

while passing in iterator form.

Nesting in MAPS

```
main()
{
    map<pair<int,int>,int> m;
    pair<int,int> P1, P2;
    P1 = {1,2};
    P2 = {2,1}
```

In unordered map, pair is not allowed.

```
Cout << (P1 < P2); => 1
Cout << (P1 > P2); => 0
```

All the pairs will be stored in sorted array.

Sets in map

```
main()
{
    map<set<int>, int> m;
```

```
set<int> s1 = {1,2,3};
set<int> s2 = {2,1};
cout << (s1 < s2); ⇒ 1
```

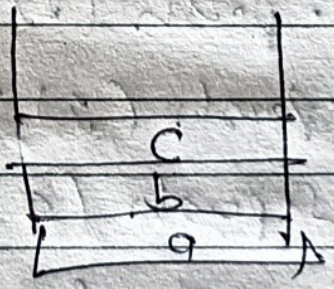
Now implem complex

```
main()
{
    map<pair<string, string>, vector<int>> m;
    int n;
    cin >> n;
    for (int i=0; i<n; i++)
    {
        string fn, ln;
        int cl;
        cin >> fn >> ln >> cl;
        for (int j=0; j<cl; j++)
        {
            int x;
            cin >> x;
            m[{fn, ln}].push_back(x);
        }
    }
}
```

```
for (auto &pr : m)
{   auto &full_name = pr.first;
    auto &list = pr.second;
    cout << full_name.first << " " <<
            full_name.second;
    cout << list.size(); << endl;
    for (auto &element : list)
    {   cout << element << " "; }
    cout << endl;
}
```

STACK (LIFO)

Stack
a, b, c



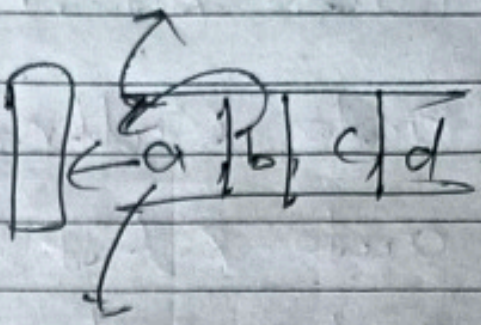
push, top, pop

Printing order :- c, b, a

Date
Page

Queue

a, b, c, d



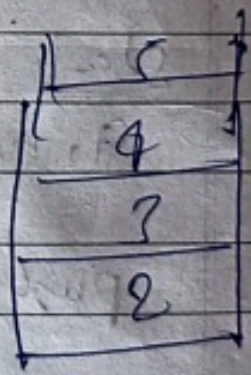
Priority order => a, b, c, d

Any data that will be passed at end pos^n.

front(), push(), pop().

Code => STACK

```
main()
{                                  → double / float
    stack<int> s;
    s.push(2);
    s.push(7);
    s.push(4);
    s.push(5);
    while(!s.empty())
```

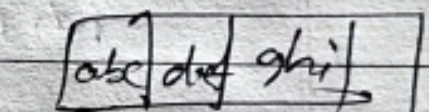


```
{
    cout<< s.top() << endl
    s.pop();
}
```

Priority Order = &, |, ^, 2

Queue

```
main()
{
    queue<string> q;
    q.push("abc");
    q.push("def");
    q.push("ghi");
    while(!q.empty())
    {
        cout<< q.top() << endl;
        q.pop();
    }
```



Priority as => abc, def, ghi

Balanced bracket matching

(( )) (✓)	{{}} (✓)
((((x)	()()(()() (x)
(() (x)	
{{}} (✓)	

Ex:- (( )) ( )

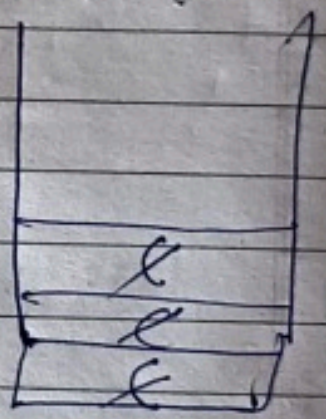
→ (( )) ( )

push push pop pop (✓) pop ✗

push ✗ ✗ push

(= opening

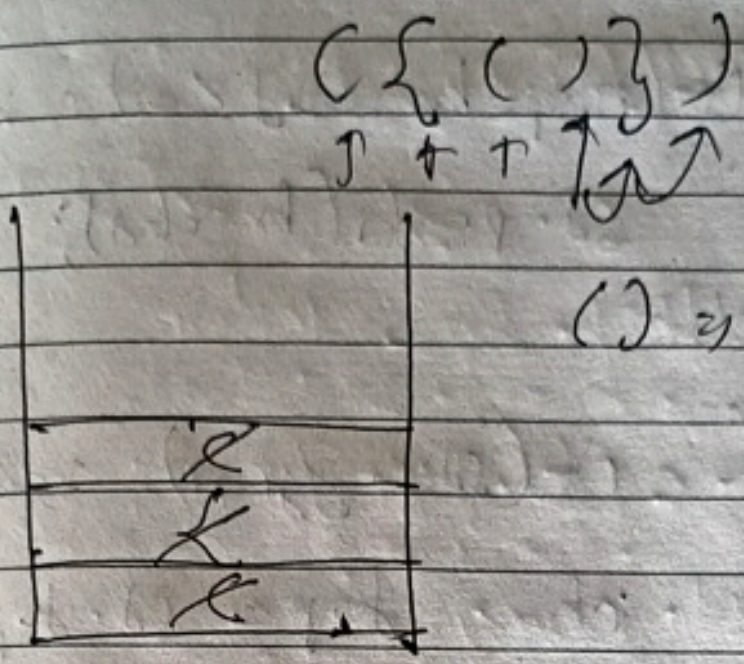
) = closing



If stack is empty then our string is balanced unless there won't be close bracket

Date
Page

Ex-1 left



Code

```
main()
{
    string s;
    cin>>s;
    if(balanced(s));
}
Unordered_map<int, char> map;
void Isbalanced(string s)
{
    Stack<char> st;
```

```
for(char bracket:s)
{
    if(map[bracket]<0)
        st.push(bracket);
    else
    {
        if(st.empty()) return "No";
        char top=st.top();
        st.pop();
        if(map[top]+map[bracket]
                !=0)
            return "No";
    }
}
if(st.empty()) return "Yes";
    else
    return "No";
```

Next Greater Element

```
main()
{
    int n;
    cin >> n;
    vector<int> v(n);
    for(int i=0; i<n; i++)
    {
        cin >> v[i];
    }
    vector<int> nge = NGE(v);
    for(int i=0; i<n; i++)
    {
        cout << v[i] << " " << v[nge[i]]
    } cout << v[i] << " " << (nge[i]==-1 ? -1
                              : v[nge[i]]);
vector<int> NGE(vector<int> v)
{
    vector<int> nge(v.size());
    stack<int> st;
```

```
for (int i=0; i<v.size(); ++i)
{ while(!s.empty() && v[i]>
                    v[st.top()])
  { //st.top = index
    nge[st.top()] = i;
    st.pop()
  }
  st.push(i);
}
while(!s.empty())
{ nge[st.top()]=-1;
  st.pop();
}
return nge;
}
```

// nge[i] = index of next greater.

Inbuilt C++ STL

```
main()
{ int n;
  cin>>n;
  int a[n];
  for (int i=0; i<n; i++)
  {     cin>>a[i]; }
  sort (a, a+n);
        ↑     ↑
   initial   address of one
   start of  more point of
   sort      our sort
  for (int i=0; i<n; i++)
  {     cout<<a[i]; }
  sort (a+2, a+n)
  → a, a+1, a+2, a+3, a+... a+n
```

Introsort contains three algorithm like quick sort, heap sort & insertion sort.

Incase of vector

```
vector<int> a(n);
sort(a.begin(), a.end())
```

(7) Comparator function

```
//swap(a,b); it is a inbuilt fun
but for normal sorting.
for(i=0; i<n; i++)
{ for(int j=i+1; j<n; j++)
  { if(a[i]>a[j])
    { swap(a[i], a[j]); }
```

replace

a[i] > a[j] by

if(should_i_swap(a[i], a[j]))

{ swap(a[i], a[j]); }

should_i_swap(int a, int b)

{ if(a > b) return true;

return false;

}

In case of pair

vector<pair<int, int>> a(n);

Algo for(int i=0; i<n; i++)

{ for(j=i+1 → n)

{ if(should_i_s(a[i], a[j])

swap(a[i], a[j]);

} }

```
should_i_swap(pair<int,int>a,
              pair<int,int>b)
{
  if( a.first != b.first)
  {
    if (a.first > b.first)
      return true;
    return false;
  }
  else
  {
    if (a.second < b.second)
      return true;
    return false;
  }
}
```

sort(a.begin(), a.end()); // by default sort

Comparator func says that
if you give false then swap
else not swap.

Use Comparator

```
main()
{
    int n;
    cin>>n;
    vector<pair<int,int>> a(n);
    for(int i=0; i<n; i++)
    {
        cin>>a[i].first>>a[i].second;
    sort(a.begin(), a.end(), comp);
    for(int i=0; i<n; i++)
    {
        cout<<a[i].first<<" "<<
            a[i].second;
}
```

```
bool cmp(pair<int,int> a, pair<int,int>
                                    b)
{
    if(a.first != b.first)
    {
        return a.first < b.first;
    }
    return a.second > b.second
}
```

for int

```
bool cmp(int a, int b)
{
    return a>b;
```

⑪ Upper Bound And Lower Bound

```
let main()
{
    int n;
    cin >> n;
    int a[n];
    for(int i=0-n)
```

```
{
    cin>>a[i];
}
sort(a, a+n);
for(int i=0; i<n; i++)
{ cout<<a[i];
```

Upper bound

Number which is greater than pointing number would be print

Lower bound

Number which is less than or equal to pointing number.

Both UB & LB produces location in arr ⇒ pointer return
vector ⇒ iterator

```
int *ptr = lower_bound(a, a+n, 5);
cout << (*ptr) << endl;
if (ptr == (a+n))
{
    cout << "Not found";
}

int *ptr = upper_bound(a, a+n, 5);
if(ptr == (a+n))
{
    cout << "Not found";
    return 0;
}
cout(*ptr);
```

In case of Vector

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

```
sort(a.begin(), a.end());
for (int i=0; i<n)
{   cout<< "Not found";
    return 0;
}
cout<<(*it)<<endl;
```

Incase of set

```
main()
{   int n;
    cin>>n;
    set<int> s;
    for(int i=0; i<n)
    {   int x;
        cin>>x;
        s.insert(x);
    }
```

or

```
for (int i=0; i<(int)(1e6); ++i)
{
    s.insert(rand());
}
for (int i=0; i<(int)(1e5); ++i)
{
    auto it = s.lower_bound(rand());
}
```

In case of map, make yourself

(#) C++ Inbuild Algorithm

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

① min_element(v.begin(), v.end())

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

cout << min;

② max_element(v.begin(), v.end())

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

③ Accumulate means sum.

accumulate(v.begin(), v.end(), 0)

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

④ count

int ds = count(v.begin(), v.end(), 7)

or any num

(5) (find) ele

it finds elem in array & return pointer or iterator

```
int ele = * find(v.begin(), v.end(), 2);
```

or

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

(6) reverse

```
reverse(v.begin(), v.end())
for (auto val : v)
{   cout val }
```

In case of string

```
string s2 "abcdefghi";
reverse(s.begin(), s.end());
cout << s << endl;
```

or

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

⑦ Inbuilt STL Algo & Lambda fun

```
main()
{
    cout << [](int n){ return n+2; }(5)
         << endl;
```

or

```
cout << [](int n, int y){ return n+y; }
        (4,7) << endl;
```

or

```
auto sum2 = [](int n, int y){ return n+y;
```

cout << sum(2,3);

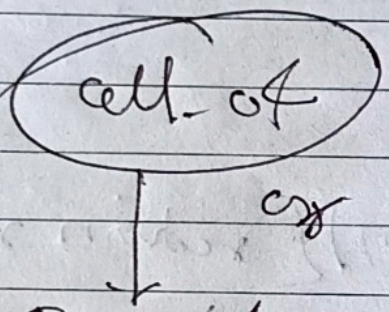
bt

main()

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

cout << all_of(v.begin(), v.end(),

[](int n){return n>0;});



func check true or false

This func return true if all ele of vector are true.

or

cout << all_of(v.begin(), v.end(),

is_positive);

bool Is_positive (int n)
{ return n>0; }

② Any_of

It returns true if atleast one elem returns true.

Cout << any_of(v.begin(), v.end(),
[](int n){return n>0;});

③ None_of

It returns true if all ele are -ve.