STL : Standard Template libraries [C++]

```
#include <bits/stdc++.h>          [we use this instead of writing all libraries like string.h, math.h separately]
using namespace std;
```

```
pair <int, int > p : {1,3} ;                    # p = {1,3}

cout << p. first << "  " << p. second ;   # 1 , 3
```

nested property of pair

```
pair<int, pair<int, int>> p = { 1, {3, 4}} ;

Cout<< p.first << " " << p. second. second << " " << p. second. first;

                    # 1  4  3
```

pair can be a data type , lies inside utility library

```
pair <int, int > arr [ ] = { {1,2} , {2,5}, {5,1}};
cout << arr [1]. second ;   ⟶ #5
```

vector

(arrays are constant in size)

↓

containers : dynamic in nature

vector < int > v; ⟶ creates an empty container { }

v.push_back (1); ⟶ {1}

v.emplace_back (2); ⟶ {1, 2} # faster than push_back

vector < pair <int, int>> vec;

vec.push_back ({1,2}); ⟶ isme we have to mention curly brackets for it to be considered as pair.

vec.emplace_back (1,2);

↓

isme not reqd.

vector <int> V(5,100); ⇒ {100, 100, 100, 100, 100} (5 instances of 100)

0 1 2 3 4

vector <int> v(5); 5 garbage value in the vector.

vector <int> v1(5,20);
vector <int> v2(v1); → v2 will be another container which will be copy of v1.

v1.push_back(1); {20,20,20,20,20,1} #allowed (dynamic nature)

* accessing elements in vector

v → {20, 10, 11, 12, 13}

v[1] → 10
v[3] → 12
or v.at(1) → not used much

* iterators

Syntax: datatype :: iterator x = v.begin()
vector <int> :: iterator it = v.begin()
it++;

points directly

{20, 10, 15, 6, 7}

cout << *(it) << " " ; → (10) ← to memory

it = it + 2;

cout << *(it) << " "; → (6) ←

vector<int>:: iterator it = v.end() {10, 20, 30, 40}

↳ points to a memory location right after last element, not 40.

vector<int>:: iterator it = v.rend() - {10, 20, 30, 40}

reverse end

vector<int>:: iterator it = v.rbegin() {10, 20, 30, 40}

↓

it++ : 30

cout << v.back() << " " ; {10, 20, (30)}

*printing the vector:

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

{10, 20, 30}

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

for (auto it:v) {
        cout << *(it) << " ";
}
```

(for each loop)

* deletion in vector:

v.erase (v.begin()+1); {10,20,12,13} → {10,12,13}

v.erase (v.begin()+2, v.begin()+4); # [start, end)

↳ not included

start ↓ included

{10, 20, 12, 23, 35} → {10, 20, 35}

* insert function:

vector<int>v (2, 100); {100, 100}

v.insert (v.begin(), 300); {300, 100, 100}

v.insert (v.begin()+1, 3, 10); {300, 10, 10, 10, 100, 100}

position no. of occurences

vector <int> copy (2, 50); // {50, 50}

v.insert (v.begin(), copy.begin(), copy.end());

// {50, 50, 300, 10, 10, 10, 100, 100}

// {10, 20}

cout<<v.size(); → 2

v.pop_back(); → {10} (pops out last element)

// v1: {10, 20}, v2: {30, 40}

v1.swap(v2); // v1 = {30, 40} v2 = {10, 20}

v.clear ; → erases everything

cout << v.empty() ; // true or false.
