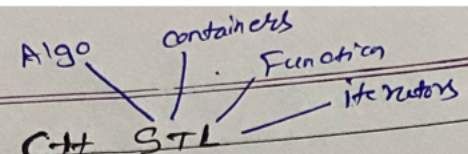


Date

(1) using namespace std;
↝ You don't need to write (std::) everywhere if you use this

(2) #include <bits/stdc++.h>
↝ you don't need to include <string.h>, <math.h> etc separately. this will import all essential libraries

(3) pairs
You can store more than one value in a single variable

pair <int, int> p = {1, 3};
here p has 1 and 3 both value
now, to access them separately

cout << p.first << " " << p.second

↝ How to store more than 2 values
pair <int, pair <int, int>> p = {1, {3, 4}};
to access them
cout << p.first << " " << p.second.first << " " << p.second.second

↝ declaring a pair array
pair <int, int> arr[] = {{1, 2}, {2, 5}, {5, 1}};
to access them
cout << arr[1].first/second

Vector

type of container which stores element ~~element~~ in a similar fashion as array does but we can change its size whenever we want unlike array

How to declare a vector

```
vector<int> v;
```

v.push_back(1); ⇝ creates an empty container and stores 1 in it

v.emplace_back(2); ⇝ dynamically increases the vector size and place 2 in back

to declare in pair type (name of variable)

```
vector<pair<int,int>> vec;
vec.push_back({1,2});
vec.emplace_back(1,2);
```

to access any element

```
cout << v[0] << " ";
```

{20,10,15,6,7}

by iterator

Syntax of iterator ⇝ points to the memory address

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

basically pointing to the memory

```
cout << any_name;        ⇝ print the address
cout << *(any_name);     ⇝ print the value
```

others iterators

v.end(); ⇝ it will point to something which was beyond the last element, which we don't know (right after last element)

v.rend(); ⇝ points just before first element

v.rbegin(); ⇝ points to last element

if we do increment then it will point second last element

(x) to print all element of vector

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

now this is long form, so we also use

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

(x) to erase/delete element in a vector

```
v.erase(v.begin()+1);
```
{10, 20, 12, 23}

for more than one element
{10, 20, 12, 23, 35}

```
v.erase(v.begin()+2, v.begin()+4);
```
↳ output = {10, 20, 35}

[start, end)

hume jo element ko erase krna hai
hamesa usse ek aage waale element
ki position deni hai.

(x) insert function

```
vector<int> v(2, 100);   ⇝ {100, 100}
v.insert(v.begin(), 300);   ⇝ {300, 100, 100}
v.insert(v.begin()+1, 2, 10);   ⇝ {300, 10, 10, 100, 100}
```

(x) copy

```
vector<int> copy(2, 50);   ⇝ {50, 50}
v.insert(v.begin(), copy.begin(), copy.end());
```
↳ {50, 50, 300, 10, 10, 100, 100}

we can do front operations here

Page No.
Date

⊗ List

a container which is dynamic in nature

```
list <int> ls ;

ls.push_back(2); ⇝ {2}
ls.emplace_back(4); ⇝ {2,4}
ls.push_front(5); ⇝ {5,2,4}
ls.emplace_front(); ⇝ {2,4}
```

* rest func^n same as vector
* begin, end, rbegin, rend, clear, insert, size, swap

⊗ Deque

exactly similar to list and vectors

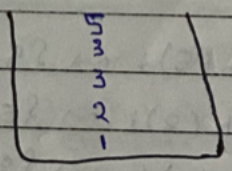
⇝ func^n are same as vector like begin, end etc.

⊗ Stack ⇝ LIFO

only 3 func^n here
- push
- pop
- top

Declaration ⇝

```
Stack <int> st ;
st.push(1); ⇝ {1}
st.push(2); ⇝ {2,1}
st.push(3); ⇝ {3,2,1}
st.push(3); ⇝ {3,3,2,1}
st.emplace(5); ⇝ {5,3,3,2,1}
```



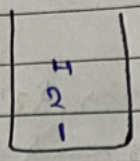
* here indexing is not done so we cant print element with st[2] etc.

```
cout<< st.top(); ⇝ print 5

st.pop(); ⇝ {3,3,2,1}
cout << st.top(); ⇝ print 3
cout << st.size(); ⇝ 4
cout << st.empty(); ⇝ False
```

⊗ Queue ⇝ FIFO

Declaration ⇝ queue <int> q;



q.push(1); ⇝ {1}
q.push(2); ⇝ {1,2}
q.emplace(4); ⇝ {1,2,4}

cout << q.back() += 5; ⇝ print 9
Queue is {1,2,9}

cout << q.front(); ⇝ prints 1
q.pop(); ⇝ {2,9}

cout << q.front(); ⇝ prints 2

* size, swap, empty func^n same as stack

⊗ priority_queue
larger the number, more will be its priority
declaration ⇝ priority_queue <int> pq;

pq.push(5); ⇝ {5}
pq.push(2); ⇝ {5,2}
pq.push(8); ⇝ {8,5,2}
pq.emplace(10); ⇝ {10,8,5,2}

cout << pq.top(); ⇝ prints 10
pq.pop(); ⇝ {8,5,2}
cout << pq.top() ⇝ prints 8

If we want to do it in reverse order

priority_queue<int, vector<int>, greater<int>> pq;
pq.push(5); ⇝ {5}
pq.push(2); ⇝ {2,5}
pq.push(8); ⇝ {2,5,8}
pq.emplace(10); ⇝ {2,5,8,10}

cout << pq.top(); ⇝ prints 2

(8) Set

Set ⇝ Sorted order
⇝ unique

declare ⇝ set<int> s;

s.insert(1); ⇝ {1}
s.emplace(2); ⇝ {1,2}
s.insert(2); ⇝ {1,2}
s.insert(4); ⇝ {1,2,4}
s.insert(3); ⇝ {1,2,3,4}

{1,2,3,4,5}

auto it = s.find(3)

when it don't find the element it prints st.end();

auto it = s.find(6) ⇝ st.end()

s.erase(5); ⇝ erase 5 and maintains sorted order

⊗ Multiset ⇝ Also sorted

Same as set but can store duplicate element

```
multiset<int> ms;
ms.insert(1); ⇝ {1}
ms.insert(1); ⇝ {1,1}
ms.insert(1); ⇝ {1,1,1}

ms.erase(1); ⇝ all 1's erased

ms.count(1);
```

To erase only a single 1

```
ms.erase(ms.find(1));
```

rest all func same as set

⊗ Unordered set

not sorted but unique ↗ randomised order

TC ⇝ O(1)

★ All func are same as of set but lower_bound and upper_bound don't work here

⊗ map

stores element like → key (unique) value (can be duplicate)

1	~
2	~
3	~
4	~
⋮	

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

mp[1]=2;
```

map Stores key in Sorted order

Page No.
Date

key	value
1	2
2	4
3	1

```
mp[1] = 2;
mp.emplace({3,1});
mp.insert({2,4});
```

{1,2}
{2,4}
{3,1}

```
mp[{2,3}] = 10;
```

~> key {2,3} value 10