C++ Basics DSA:

1. #include <iostream> ⟶ to take I/P (or) O/P.
   #include <math.h> ⟶ math fun's.
   <string> ⟶ string fun's.

2. print O/P:

```
#include <iostream>
using namespace std;
int main() {
    cout << "Siva";
    cout << "Mani";
    return 0;
}
```

→ O/P : Siva Mani

if : `cout << "Siva" << endl;`
`cout << "Mani";`

O/P: Siva
Mani.

```
#using namespace std;
```

3. Take input:

```
int main() {
    int x;
    cin >> x;
    return 0;
}
```

way 1:

```
int x,y;
cin >> x >> y;
```

Take I/P in the same line.

i.e 10 20

way 2:

```
cin >> x;
cin >> y;
```

i.e 10
20

4. Include all libraries ✓

```
#include <bits/stdc++.h>
```

for every prog: write.

```
#include <iostream>
#include <bits/stdc++.h>

using namespace std;
```

Datatypes:

Integers:

1. int
2. long
3. long long

Decimal:

1. float
2. double

Ex: float x=20;
cout<<x;
O/P: 20

String:

1. string
2. getline

Ex:
```
string s;
cin >> s;
cout << s;
```
I/P: Hey Hi Siva

O/P: Hey

getline → whole sentence / line

Ex:
```
string str;
getline(cin, str);
cout << str;
```
I/P: Hey Siva

O/P: Hey Siva

char:

Ex:

```
char ch;
cin >> ch;
cout << ch;
```

→ single quotes ' '

Ex: `char ch = 'A';`

~~Ex:~~ string str = "A";

---

condnal stmts:

if / if-else / if-else if -else

1. only if
2. else with only if
3. if — else if

sometime no need of "else".

nested if else:

```
if (condn)
{
    if(cond) {
    }
    else if (condn)
    }
    else {
    }
else {

    if
    else if
    else
```

---

unsigned

↓

0 → +ve numbers.

Ex:

⇒ int → -2 million to 2 million

(or) signed int

unsigned int → 0 to 4 million.

⇒ long → $-9 \times 10^6 \times 10^6$

→ 9×10^6 million ($\times 10^6$)

unsigned long → 0 to 18×10^{12}

int → 10^9 (-/+)

long → 10^{12} (-/+)

long long → 10^{18} (-/+)

Switch - Cases :

Ex: int *ch; // char ch;

cin >> ch;

switch (ch) {

case 1 :

:

break;

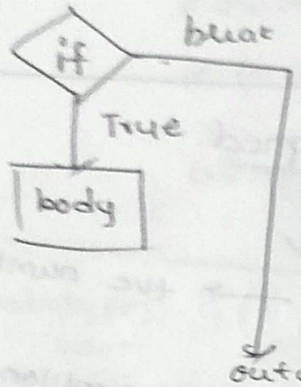
case 2 :

:

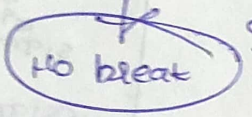
break;

:

default : cout << "Enter valid ch



with / with out body stmt..



Ex:

default :

(or)

default : cout << "X";

Data Structures

- ~~Static~~ Linear
  - Static
    - ⇒ Arrays
  - Dynamic
    - ⇒ Linked lists
    - ⇒ Stack
    - ⇒ Que
- Non-linear
  - ⇒ Tree
  - ⇒ Graph

Arrays:

way 1:

```
int main () {
    int n;
    int arr[n];
    cin >> n;
```

array can be
- int
- double
- string.

---

functions:

```
datatype fun_name ( parm_list) {
        body;
        return stmts;
}
int main () {
    fun_name ( parm);
    return 0;
}
```

Ex:

```
void print (string name){
    cout<<"Hey"<<name;
}
int main (){
    string str="Siva";
    print (str);
    return 0;
}
```

O/p: Hey Siva.

1. pass by value

↓

Not to change original value

⇒ send only the copy of original.

2. pass by reference

↓

change original value

Ex:-

```
void print(string s){
    s[0]="t";
    cout << s <<endl;
}
```

```
int main(){
    string s="Raj";
    print(s);
    cout << s <<endl;
}
```

o/p:

Taj

Raj

```
void print(string &s);
    s[0]='t';
    cout <<s <<endl;
}
```

o/p:

Taj

Taj

Arrays:

↳ => Should always pass by Reference.

i.e dtype fun(int arr)
name

=> When passed in a fun

array will get modified.

Week 1:

↳ Compute Arrays

Ex:

```
void Print (int arr, int n){
    arr[0] += 100;
    cout << arr[0];
}

int main(){
    int n = 5
    int arr[n];
    print(arr, n);
    cout << arr[0];
}
```

O/P:

inside fun : 105

in main : 105

c++ STL:

Set

1. Unordered set
2. vector
3. set
4. unordered multiset
5. multiset

Map

6. map
7. unordered map
8. multimap
13. unordered multimap

Stack/queue

9. queue
10. priority queue
11. dequeue
12. stack

List

14. list
15. next permutation

20. pair

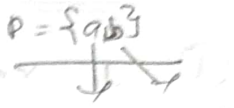
Builtin fun^ns

16. builint_popcount()
17. sort()
18. min element()
19. max element()

C++ STL :

1. Algorithms
2. containers
3. functions
4. Iteratols.

1. Pairs: pair <int, int> p = {1, 3};



To access elements:

cout << p.first ⟶ 1

cout << p.second ⟶ 3

Pair <a, b> obj = {A, B}

a, b → datatype

A, B → values.

Ex 2:

pair < int, pair < int, int > > p = {1, {2, 3}};

=> p.first ⟶ 1

p.second ⟶ {2, 3}

p.second.first ⟶ 2

Ex 3: Pair array

pair <int, int> arr [] = { {1, 2}, {3, 4}, {4, 5} };

arr[0] arr[2].

=> arr[1].first ⟶ 3

arr[1].second ⟶ 4

2. Vectors → container, dynamic (size can be increased)
→ work like array

Syntax: vector <int> v ; → v = { }

operations:

(A) push_back
1, v.pushback (1) → v = {1}

3, v.emplace_back (2) → v = {1,2}

(B) vector < pair <int, int> > vec ;

⇒ vec.push_back ({1,2}); → vec = { {1,2} }

⇒ vec.emplace_back (3,4); → vec = { {1,2}, {3,4} }

(C)
1. vector <int> (v)(5,100) → v = {100,100,100,100,100}
size, value

2. vector <int> v (5) → v = {0,0,0,0,0}

3. vector <int> v1 (5,20) → v1 = {20,20,20,20,20}

ii. v2(v1) → v2 = {20,20,20,20,20}

v1 {1,2}

v2(v1)

v = { }

v.push_back ({1,2})

emplace_back (1,2)

{1,2}

Accessing Elements in — Vectors:

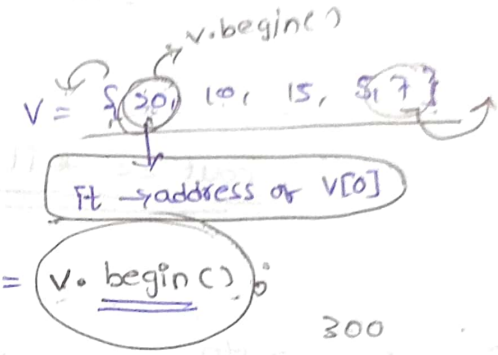
v → {20, 10, 15, 5, 7}

v[0] = 20 v.at(0) → 20

v[4] = 7

v[0] ↓ v.at(0)

⇒ Iterators



Syntax:

vector<int> :: iterator it = v.begin(); 300

⇒ cout << *(it); → 20

it ++;

cout << *(it); → 10.

it → *it

it → v.end()

it--

*it

V = { 5, 10, 15, 20 }

Types:

(A) vector<int> :: iterator it = v.begin() → *(it) = 5

v.end() ✓ *(it) = 20 wrong

v.rend() ✓ *(it++) 5

v.r begin() ✓ *(it) = 20

it points after 20,

*(it)-- → 20

To print all elements in a vector:

$V = \{5, 10, 15, 20\}$

① for (vector<int>::iterator it = V.begin() ;
it != V.end() ;
it ++) {

cout << *it << " " ; → 5 10 15 20

② using auto → iterators

for (auto it = V.begin() ;
it != V.end() ; it ++) {

cout << *it << " " ; → 5 10 15 20

③ auto

for (auto it : V) {

cout << it << " " ; → 5 10 15 20

DELETION in vector:

1. V.erase (V.begin()+1) → V = { 5, 10, 15, 20 } → V = { 5, 15, 20 }

2. V.erase (V.begin()+2 , V.begin()+4)

start end

i = 2 j = 4

V = { 1, 2, 3, 4, 5, 6 }

V = { 1, 2, 5, 6 }

i j

[ )

INSERT in vector:

vector<int> v(2,10) ⟶ v = {100,100}.

1.

(A) v.insert(v.begin(), 300) ⟶ v = {300,100,100}

(B) v.insert(v.begin()+1, 2, 10) ⟶ v = {300,10,10,100,100}

2. Insert vector into vector.

v1 = {10,20} v2 = {300,100,100,200}

(A) v2.insert(v2.begin, v1.begin(), v1.end())

[ )

⟶ v2 ⇒ {10,20,300,100,100,200}

size() v1 ⟶

other functions:

v = {10,20,30,40}

1. size ⟶ cout << v.size(); ⟶ 4

2. pop_back ⟶ cout << v.pop_back(); ⟶

v = {10,20,30}

v1 v2

3. swap ⟶

v1 = {10,20} v2 = {30,40}

v1.swap(v2); ⟶ v1 = {30,40}

v2 = {10,20}

4. clear ⟶

v.clear() ⟶ v = { }

5. empty() ⟶ true/false

Ex: v = {1,2}

cout << v.empty(); ⟶ false

3. List ⟶ similar to vector.

Syntax: list<int> ls;

operations:

ls = {1,2,3,4}

i, push_back ⟶ ls.push_back(5) → ls = {1,2,3,4,5}

ii, emplace_back ⟶ ls.emplace_back(6) → ls = {1,2,3,4,5,6}

iii, push_front ⟶ ls.push_front(10) → ls = {10,1,2,3,4,5,6}

iv, emplace_front ⟶ ls.emplace_front(20) → ls = {20,10,1,2,3,4,5,6}

NOTE: rest of all functions are same as vector

like begin(), end(), rend(), rbegin(), insert(), pop_back(), swap(), erase(), clear(), size(), empty()

---

4. Deque / deque

Same as List

⇒ All list functions will work.

Syntax:

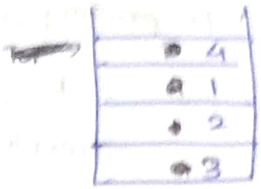
deque<int> dq;

dq = {1,2,3,4,5}

1. dq.front(); ⟶ 1

2. dq.back(); ⟶ 5

5. stack : ⟶ LIFO



Syntax: stack<int> st ;

operations: st = {1,2,3}

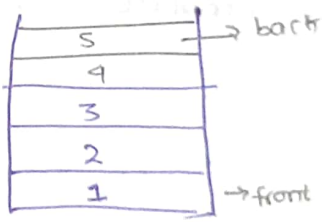
1. push() ⟶ st.push(4) ⟶ st = {4,1,2,3}
   st.push(6) ⟶ st = {6,4,1,2,3}
2. pop() ⟶ st.pop() ⟶ st = {4,1,2,3}
3. top() ⟶ st.top() ⟶ (4)

4. size() ⟶ st.size() ⟶ 4
5. empty() ⟶ st.empty() ⟶ st = { }
6. swap()

```
stack<int> st1,st2;
st1.swap(st2);
```

---

6. Queue : ⟶ FIFO



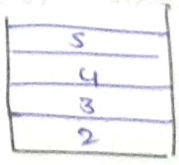
Syntax:

queue<int> q ; q = {1,2,3}

1st to enter comesout 1st

operations:

1. push() → q.push(4) ⟶ q = {1,2,3,4}
2. emplace() → q.emplace(5) ⟶ q = {1,2,3,4,5}
3. q.front() ⟶ 1
4. q.back() ⟶ 5
5. q.pop() ⟶ q = {2,3,4,5}



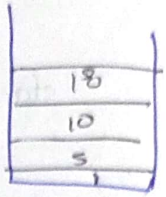
// size, swap, empty same as stack.

7. priority_queue : (max-heap)

↓

elements enter in queue based on its size

large / small



⇒ Same queue operations.

push } → log(n)
pop
top → O(1)

..

8. set

- sorted
- Unique elements.

Syntax: `set<int> s;`

operations: (log(n))

`s = { }`

i. insert → `s.insert(4)` → s = {4}

ii. emplace → `s.emplace(2)` → s = {2,4}

`s.insert(5)` → s = {2,4,5}

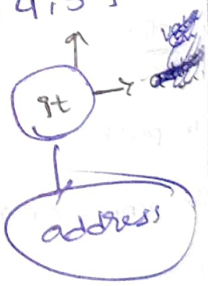
`s.insert(1)` → s = {1,2,4,5}

Iterator:

⇒ `auto it = s.find(5);`

· it points to 5.

s = {1,2,4,5}



⇒ `auto it = s.find(6);`

↓

6 not in ~~list~~ set

so, it points to s.end()

$S = \{1,2,3,4,5\}$

iii) erase → Ⓐ S.erase(5)

→ $S = \{1,2,3,4\}$

Ⓑ ⇒ auto it = s.find(4);

s.erase(it); → $S = \{1,2,3\}$

Ⓒ ⇒ $S = \{2,3,5,7,9\}$

auto it1 = s.find(3);

it2 = s.find(9);

s.erase(st1, st2) → $S = \{2,9\}$

[start, end)

iv) count :

int cnt = s.count(1) → 1

NOTE: begin, end, rend, rbegin

empty, swap → same as list

v) lower_bound, upper_bound

9. Multi set → same as set

↓

Store duplicates. → sorted, Not unique

Syntax: multiset <int> ms;

operations: All set operations.

ms = {1,2}

1. insert → ms.insert(1) → ms = {1,1,2}

2. erase → ms = {1, 1, 1, 2}

(A) ms.erase(1) → ms = {2}

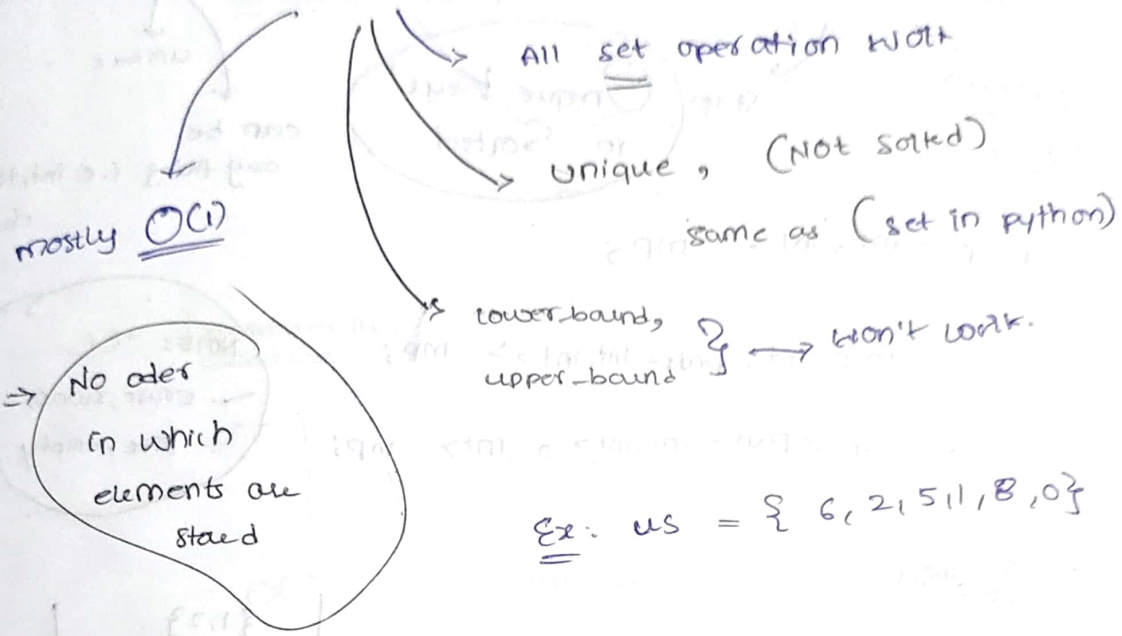
(B) ms.erase(ms.find(1)) → ms = {1, 1, 2}

(C) ms.erase(ms.find(1), ms.find(1)+2); → ms = {1,2}

3. count → ms = {1,1,1,2}

ms.count(1) → 3

10. Unordered set



Ex: us = {6, 2, 5, 1, 8, 0}

Syntax: unordered set<int> us;

Worst case : O(n)

---

11. Map

map => stores {key: value}

key → unique

O(log N)

syntax:

stores Unique Keys in Sorted

can be any thing i.e int, float, char, double

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

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

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

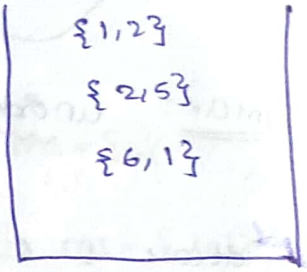
NOTE:
erase, swap, size, empty

How it stores:

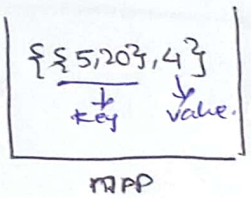
i. `mp[1] = 2;`

`mp.insert({2,5})`

`mp.emplace({6,1})`



ii. `mpp[{5,20}, 4]` ⟶



Iterator:

map = [[1,2] [4,9] [32,12]]
it it it
value

① `for(auto it : mpp) {`

`cout << it.first` ⟶ keys => 1, 4, 32

// `cout << it.second` ⟶ values => 32, 9, 12.

② `auto it = mp.find(3);`

mp = [[1,4], [3,10]]
it
(address)

`*(it).first` ⟶ 3

`*(it).second` ⟶ 10

12. Multimap

- Same as mp
- can store duplicate keys
- mp[key] → can't be used.

13. Unordered Map:

- unique key
- NOT sorted

↓

O(1) worst O(N)

ALGORITHMS:

1. sorting:

i) sort (a, a+4) → arrays

↓ ↓

array a+n

↓ size.

[start, end)

ii) Vectors → sort (v.begin(), v.end())

=> arr = {1,3,5,2}

//sort in decending order.

- sort (a, a+n, greater<int>) ; → arr = {5 3 2 1}