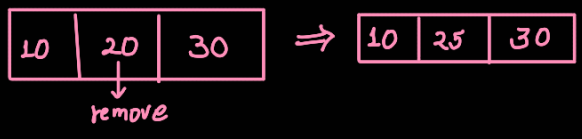
[SETS]

→ Implemented using Binary Search trees (BST)
⇓ more precisely
Red-black tree

#1. STL container → stores unique elements.
2. Values are stored in ordered state (increasing / decreasing order)
3. No indexing, elements are identified by their own values.
4. Once value is inserted in a set, it cannot be modified.
⇓
pehle purani value remove karo,
then new value daalo.



Advantages:
1. to store unique values.
2. store data in ordered manner.
3. Dynamic size.
4. Faster → insertion, deletion, searching } O(log N) time complexity
↓
binary search

Disadvantages:
1. Cannot access elements using indexing.
2. consumes more memory.
3. not suitable for large data-sets.
↳ insertion/deletion : O(log N)

declaration:
```
#include <set>
set <data_type> set_name;
```
eg → `set <int> set1;`
initialise → `set <int> set2 = {1, 3, 2, 4};`

by default, values are stored in increasing order

∴ set2 = {1, 2, 3, 4} // done automatically.

to store values in a set in decreasing order:

```
set <int, greater <int> > set2;
```

Insertion in set ⇒

```
set1.insert (4);
set1.insert (1);
```

} [1, 4] set1

↓

this func" returns an iterator to the inserted value.

pointers ←

[Traversal of a Set]:

using → iterator

- set_name.begin( ); → iterator pointing to 1st element of my set.
- set_name.end( ); → iterator pointing to the pos" after the last element of my set.

SetA: [1, 2, 3, 4]

↑ setA.begin( )

↑ setA.end( )

```
set <int> set1 = { 4, 1, 5, 3};
Set1 <int>:: iterator itr; // or auto itr = set1.begin( );

for (itr = set1.begin( ); itr != set1.end( ); itr++){
    cout << *itr << " ";
}
```

O/P → 1 3 4 5

deleting → 1. set1.erase(3);
2. set1.erase(itr);
} → O(log N)

set_name.erase(value / posn); (posn → iterator)

3. set_name.erase(start pos, end pos);

deletes elements from start posn including it till end posn, excluding it. i.e. [start, end)

eg → set → [1, 2, 3, 4, 5] (s → 2, e → 3)

s.erase(s, e) → [1, 4, 5]

O(N)

N = no. of elements in the range.

to move iterator by k steps ⇒ advance(iterator_name, k);

[MEMBER FUNCTIONS]:

→ size(), max_size() → returns max. no of elements, set can hold.

→ empty() → returns true if empty else false.

→ clear() → removes all elements from set.

→ find() → returns pos. of element if present, else returns end iterator.

→ count() → returns no. of occurrences of an element.

agar koi value hogi → return 1

if not present in set → return 0

→ lower_bound → returns element if present, else returns just greater value.

eg → S = [10, 20, 30, 40]

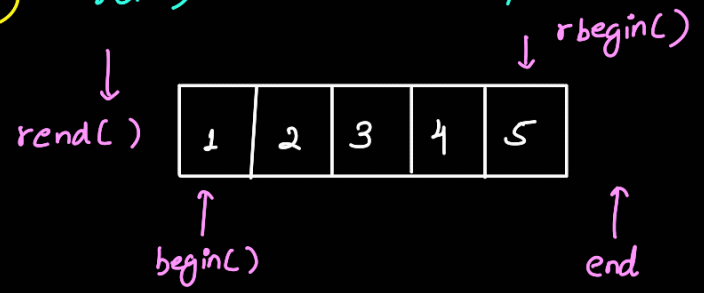
S.lower_bound(25); ⇒ returns 30.

→ upper_bound → returns next greater value.

eg → s.upper_bound(20); ⇒ returns 30.

→ rbegin() → returns iterator to first element of set in reverse order.

→ rend() → returns iterator to posn after last element in reverse order.



Unordered set : → values are stored in unordered fashion.
→ rest same as sets.

implemented using hashing

advantage over ordered set :

insertion, deletion, search } O(1) avg. time complexity : as compared to O(log N) in ordered set.

1. Insert / deletion / find / count → single element → O(1) avg.
→ O(N) worst → rehashing ← when size > capacity.

avg. case :

multiple elements : O(n) → n is no. of elements being inserted.

worst case : O(n * (N+1))
↳ size of unordered set. (rehashing)

hashing : load factor

load_factor() → $\frac{\text{size of unordered set}}{\text{bucket\_count}}$

⇒ rehash(x) → sets the no. of buckets to x or more.

(O(N) avg.)

(O(N) worst)

Multiset: It can store duplicate values. (ordered manner)

⇓

rest, same as ordered set.

Member functions:

1. insert( ) → O(log N)
2. Deletion → erase( )
   - → erase (value) → O(log N) → deletes all instance of that value.
   - → erase (position) → deletes the value at that iterator/position.
   - → erase (start, end) → O(n) → range
3. find( ) → O(log N) → lower bound of element if found else end iterator.
4. count( ) → returns no. of occurrences. O(K + log N) → no. of occ. of the element
5. upper_bound( ) and lower_bound( ) → same as set. O(log N)
6. Unordered multiset: allows duplicate values to be stored in unordered manner.
   - → member functions same as unordered_set and similar time complexities.