Introduction to STL Sets in C++

1. What is an STL Set?

- `std::set` is an associative container in C++ that stores unique elements in sorted order.
- Internally implemented using a balanced binary search tree (e.g., Red-Black Tree).

Example 1 (Integer Set):

```
set<int> numbers = { 5, 2, 8, 3, 5 };
// Stores: {2, 3, 5, 8} (duplicates removed, sorted order)
```

Example 2 (String Set):

```
set<string> simpsons = { "Homer", "Bart", "Lisa", "Bart" };
// Stores: {"Bart", "Homer", "Lisa"}
```

2. Advantages of STL Sets

+ **Unique Elements** – Automatically removes duplicates.
+ **Automatic Sorting** – Maintains elements in ascending order.
+ **Efficient Search & Insertion** – Average time complexity: **O(log n)**.

3. Disadvantages of STL Sets

- **No Direct Indexing** – Cannot access elements by index like vectors.
- **More Overhead** – Uses additional memory for tree structure.

4. Common STL set methods

1. Insertion and Deletion
2. Access and Search
3. Iterators
4. Size and Capacity
5. Advanced Set Operations

5. Insertion and Deletion

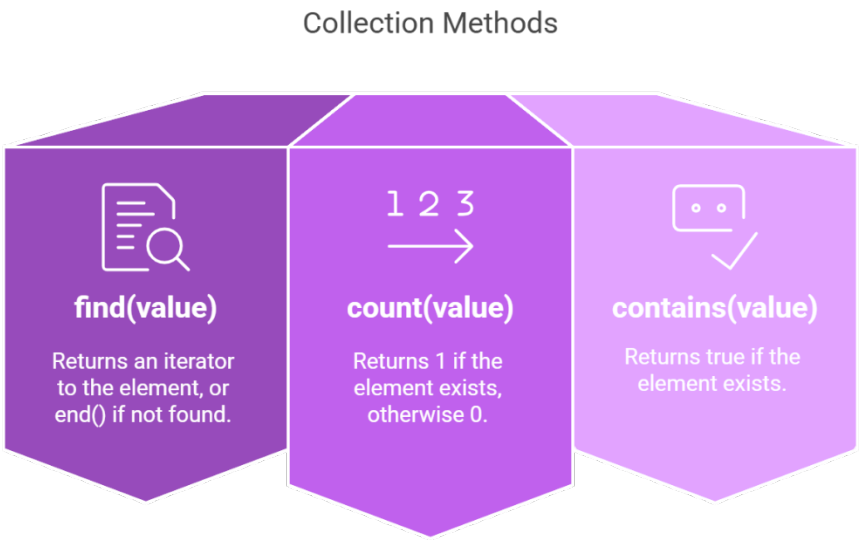
- `insert(value)` – Adds a new element (if it doesn't already exist).
- `erase(value)` – Removes an element by value.
- `clear()` – Removes all elements from the set.
- `emplace(value)` – Inserts an element (more efficient than `insert`).

Example 3:

```
set<int> numbers = { 3, 1, 4 };
numbers.insert(2);  // {1, 2, 3, 4}
numbers.erase(3);   // {1, 2, 4}
numbers.clear();    // Empty set
```

2. Access and Search

- `find(value)` – Returns an iterator to the element, or `end()` if not found.
- `count(value)` – Returns `1` if the element exists, otherwise `0`.
- `contains(value)` *(C++20)* – Returns `true` if the element exists.



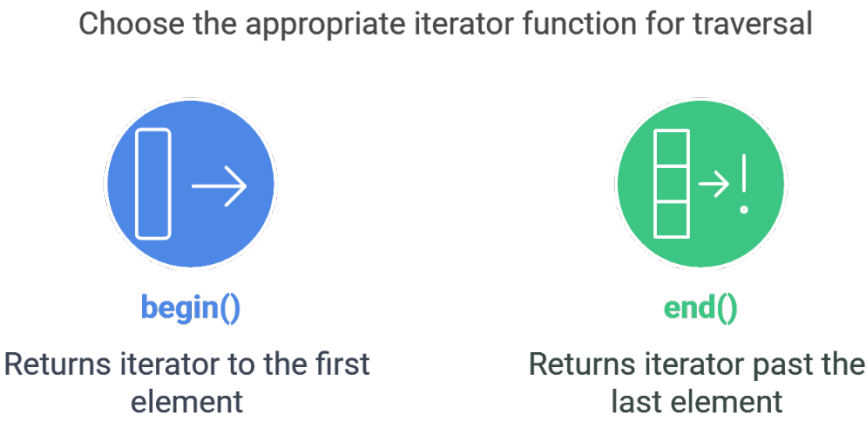
Example 4:

```
if (numbers.find(2) != numbers.end())
   cout << "2 is in the set\n";

if (numbers.count(5) > 0)
   cout << "5 exists in the set\n";
```

3. Iterators

- `begin()` – Returns iterator to the first element.
- `end()` – Returns iterator past the last element.



Example 5. Traversing a set using an Iterator

```
#include <iostream>
#include <set>
using namespace std;

int main() {
    set<string> simpsons = { "Homer", "Marge", "Bart", "Lisa", "Maggie" };

    auto it = simpsons.begin();
    while (it != simpsons.end()) {
        cout << *it << endl;
        it++;
    }

    cout << "All done!\n";
}
```

4. Size and Capacity

- `size()` – Returns the number of elements.
- `empty()` – Returns `true` if the set is empty.

Example 6:

```
cout << "Size: " << numbers.size() << "\n";
if (numbers.empty())
   cout << "Set is empty!\n";
```

5. Advanced Set Operations

- `lower_bound(value)` – Returns iterator to first element **≥ value**.
- `upper_bound(value)` – Returns iterator to first element **> value**.
- set_union
- set_difference
- set_intersection
- set_smmetric_difference

Example 7:

```
//Assume numbers holds {3, 1, 4}
auto it = numbers.lower_bound(3);  // First element ≥ 3
auto it2 = numbers.upper_bound(3); // First element > 3
```

5.1 Set Union

Here's an example of performing a **set union** in C++ using `set` and `set_union` from the `<algorithm>` header.

Example 8: Union of Two Sets (The Simpsons Characters)

```
#include <iostream>
#include <set>
#include <algorithm>

int main() {
    set<string> group1 = { "Homer", "Marge", "Bart" };
    set<string> group2 = { "Lisa", "Bart", "Maggie" };

    set<string> result;

    set_union(group1.begin(), group1.end(),
        group2.begin(), group2.end(),
        inserter(result, result.begin()));

    cout << "Union of both families: ";
    for (const auto& name : result) {
        cout << name << " ";
    }

    cout << "\nAll done!\n";
}
```

Output:

```
Union of both groups: Bart Homer Lisa Maggie Marge
```

(*Duplicates are removed automatically since `set` stores unique elements!*)