Maps in C++ STL

---

1. What is a Map?

- A `map` in C++ STL is an **associative container** that stores elements in the form of **key-value pairs**.
- Each key is unique, and it is associated with a value.
- Internally, `map` is implemented as a **self-balancing binary search tree** (typically a Red-Black Tree).
- The elements are stored in **sorted order** of keys during insertion .
- Memory Allocates on Heap
- **Heap Allocation**: Stores nodes dynamically on the heap.

---

2. Why Use a Map?

- **Key-Based Access**: Retrieve values using unique keys.
- **Automatic Sorting**: Keys are stored in ascending order by default.
- **Efficient Lookups**: Provides logarithmic time complexity (O(log n)) for insertion, deletion, and access.
- **Data Integrity**: Ensures no duplicate keys.

---

3. When to Use a Map?

- When you need to associate data uniquely with a key (e.g., dictionary-style storage).
- Examples:
    - Mapping names to phone numbers.
    - Counting occurrences of elements.
    - Caching results for efficient lookups.

---

4. How to Use a Map?

To use `map`, include the following library:

```
#include <map>
```

Declaring a Map

```
map<KeyType, ValueType> map_name;
```

- **KeyType**: Data type of the key.
- **ValueType**: Data type of the value.

Commonly Used Member Functions in Map

1. **`insert`**:
    - **Syntax**: `pair<iterator, bool> insert(pair<KeyType, ValueType> element);`
    - **Parameters**: A `pair` containing key and value.
    - **Returns**: A `pair` with an iterator to the inserted element and a boolean indicating success.
2. **`erase`**:
    - **Syntax**: `void erase(iterator pos);`
    - **Parameters**: Iterator to the element to erase.
    - **Returns**: Nothing.
3. **`find`**:
    - **Syntax**: `iterator find(const KeyType& key);`
    - **Parameters**: The key to search for.
    - **Returns**: An iterator to the element with the specified key, or `end()` if not found.
4. **`at`**:
    - **Syntax**: `ValueType& at(const KeyType& key);`
    - **Parameters**: The key to access the value.
    - **Returns**: A reference to the value associated with the key.
5. **`size`**:
    - **Syntax**: `size_type size() const;`
    - **Parameters**: None.
    - **Returns**: The number of elements in the map.
6. **`empty`**:
    - **Syntax**: `bool empty() const;`
    - **Parameters**: None.
    - **Returns**: `true` if the map is empty, otherwise `false`.
7. **`clear`**:
    - **Syntax**: `void clear();`
    - **Parameters**: None.
    - **Returns**: Nothing.
8. **`begin` and `end`**:
    - **Syntax**:
        - `iterator begin();`
        - `iterator end();`
    - **Parameters**: None.
    - **Returns**: Iterators pointing to the beginning and the end of the map.

Advantages of Using Map

1. **Sorted Order**: Ensures elements are always sorted.
2. **Efficient Operations**: Insertion, deletion, and lookups are performed in O(log n).
3. **Ease of Use**: Provides a simple interface for key-value storage and retrieval.

Disadvantages of Using Map

1. **No Duplicate Keys**: Cannot store duplicate keys (use `multimap` for that).
2. **Higher Overhead**: Compared to `unordered_map`, operations are slower due to sorting.