STL (Standard Template Library) in C++

1. Containers

- The STL provides various containers to store collections of data. Each container has unique properties and is optimized for different use cases.

- - **std::vector**: Dynamic array that allows random access. Elements are stored contiguously in memory. Best for situations where you need fast access by index and dynamic resizing.

- Example: `std::vector<int> vec = {1, 2, 3};`

- - **std::list**: Doubly linked list. Allows fast insertions and deletions from any position. Not ideal for random access, but efficient for frequent insertions and deletions.

- Example: `std::list<int> lst = {1, 2, 3};`

- - **std::map**: Sorted associative container that stores key-value pairs. Keys are unique and sorted. Ideal when you need to look up values by keys frequently.

- Example: `std::map<std::string, int> map = { {"apple", 1}, {"banana", 2} };`

- - **std::unordered_map**: Hash table-based associative container with average constant-time complexity. Keys are unique but unsorted, making it faster than `std::map` for quick lookups where order is not important.

- Example: `std::unordered_map<std::string, int> uMap = { {"apple", 1}, {"banana", 2} };`

2. Iterators

- Iterators provide a way to traverse containers, offering a generic way to access elements.

- - **begin() and end()**: Standard iterators that point to the first and one-past-the-last elements. Useful for loops.

- Example: `for (auto it = vec.begin(); it != vec.end(); ++it) { std::cout << *it; }`

- - **rbegin() and rend()**: Reverse iterators for traversing containers in reverse order.

- Example: `for (auto it = vec.rbegin(); it != vec.rend(); ++it) { std::cout << *it; }`

- - **cbegin() and cend()**: Const iterators that don't allow modification of elements.

- Tip: Use const iterators to ensure data integrity when you only need to read from the container.

3. Algorithms

- The STL provides numerous algorithms that work with iterators to perform common operations efficiently.

- - **std::sort**: Sorts a range of elements in ascending order by default. Requires random-access iterators.

- Example: `std::sort(vec.begin(), vec.end());`

- - **std::find**: Searches for an element in a range and returns an iterator to the first match or `end()` if not found.

- Example: `auto it = std::find(vec.begin(), vec.end(), 5);`

- - **std::accumulate**: Computes the sum (or another operation) of elements in a range. Requires an initial value.

- Example: `int sum = std::accumulate(vec.begin(), vec.end(), 0);`

- - **std::transform**: Applies a function to each element in a range and stores the result elsewhere.

- Example: `std::transform(vec.begin(), vec.end(), vec.begin(), [](int x) { return x * 2; });`

- Note: Familiarity with STL algorithms shows proficiency in writing efficient, concise code.