C++ Standard Template Library (STL)

1. Vectors

Purpose:

- A `vector` is a dynamic array that can grow and shrink in size. It provides random access to elements, and elements can be added or removed from the end of the vector.

Example:

cpp

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5};

    // Adding elements
    numbers.push_back(6);

    // Accessing elements
    std::cout << "First element: " << numbers[0] << std::endl;

    // Iterating through the vector
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    // Removing the last element
    numbers.pop_back();
```

```
    return 0;
}
```

Key Points:

- **Dynamic sizing**: Vectors automatically resize as elements are added.
- **Random access**: Elements can be accessed in constant time using the index operator `[]`.

2. Lists

Purpose:

- A `list` is a doubly-linked list, which allows for fast insertion and deletion of elements at any position in the list, but does not provide constant-time random access.

Example:

cpp

```
#include <iostream>
#include <list>

int main() {
    std::list<int> numbers = {1, 2, 3, 4, 5};

    // Adding elements to the front and back
    numbers.push_front(0);
    numbers.push_back(6);

    // Iterating through the list
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    // Removing elements from the front and back
```

```
    numbers.pop_front();
    numbers.pop_back();

    return 0;
}
```

Key Points:

- **Efficient insertion and deletion**: O(1) complexity for insertions and deletions at both ends and in the middle.
- **No random access**: Elements must be accessed sequentially.

3. Maps

Purpose:

- A `map` is an associative container that stores key-value pairs. Each key is unique, and the map automatically sorts its elements by keys.

Example:

cpp

```
#include <iostream>
#include <map>

int main() {
    std::map<std::string, int> ageMap;

    // Inserting key-value pairs
    ageMap["Alice"] = 30;
    ageMap["Bob"] = 25;
    ageMap["Charlie"] = 35;

    // Accessing elements
    std::cout << "Alice's age: " << ageMap["Alice"] <<
std::endl;
```

```
    // Iterating through the map
    for (const auto &pair : ageMap) {
        std::cout << pair.first << " is " << pair.second << "
years old." << std::endl;
    }

    return 0;
}
```

Key Points:

- **Associative array**: Maps store data in key-value pairs, providing fast access, insertion, and deletion based on keys.
- **Automatic sorting**: Elements are sorted by keys.

4. Sets

Purpose:

- A `set` is a collection of unique elements. It automatically sorts elements and does not allow duplicates.

Example:

cpp

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

int main() {
    std::set<int> numbers = {1, 2, 3, 4, 5};

    // Adding elements
    numbers.insert(6);
    numbers.insert(3); // Duplicate, will not be added

    // Iterating through the set
    for (int num : numbers) {
```

```
        std::cout << num << " ";
    }
    std::cout << std::endl;

    // Checking for existence
    if (numbers.find(4) != numbers.end()) {
        std::cout << "4 is in the set" << std::endl;
    }

    return 0;
}
```

Key Points:

- **Unique elements**: No duplicates allowed.
- **Automatic sorting**: Elements are stored in sorted order.

5. Algorithms

Purpose:

- STL provides a wide range of algorithms for operations like searching, sorting, counting, and manipulating collections of data. These algorithms are generic and work with any STL container.

Example:

cpp

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> numbers = {5, 2, 8, 1, 3};

    // Sorting the vector
    std::sort(numbers.begin(), numbers.end());
```

```
    // Finding an element
    if (std::binary_search(numbers.begin(), numbers.end(), 3)) {
        std::cout << "3 is in the vector" << std::endl;
    }

    // Reversing the vector
    std::reverse(numbers.begin(), numbers.end());

    // Displaying the vector
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Key Points:

- **Sorting**: `std::sort` sorts elements in ascending order.
- **Searching**: `std::binary_search` checks if an element exists in a sorted range.
- **Manipulation**: `std::reverse` reverses the order of elements.

Summary

- **Vectors**: Dynamic arrays with fast random access.
- **Lists**: Doubly-linked lists with efficient insertions and deletions.
- **Maps**: Associative containers for key-value pairs, automatically sorted by keys.
- **Sets**: Collections of unique elements, automatically sorted.
- **Algorithms**: Generic functions that perform operations like sorting, searching, and modifying data in containers.