# C++ Standard Template Library (STL)

The C++ Standard Template Library (STL) offers a variety of data structures, algorithms, iterators, and utilities to simplify coding and improve efficiency. Here’s a list of the main STL components:  
  
---  
  
 1. Containers  
Containers store collections of data. They are broadly categorized into three types: Sequence Containers, Associative Containers, and Container Adapters.  
  
# Sequence Containers  
- std::vector — Dynamic array, supports random access.  
- std::deque — Double-ended queue, supports fast insertions and deletions at both ends.  
- std::list — Doubly linked list.  
- std::forward\_list — Singly linked list (C++11).  
- std::array — Fixed-size array (C++11).  
  
# Associative Containers  
- std::set — Stores unique elements in a sorted order.  
- std::multiset — Allows duplicate elements in a sorted order.  
- std::map — Stores key-value pairs in a sorted order by key.  
- std::multimap — Allows duplicate keys in a sorted order.  
  
# Unordered Associative Containers (Hash-Based, C++11)  
- std::unordered\_set — Stores unique elements with fast access through hashing.  
- std::unordered\_multiset — Allows duplicates with fast access through hashing.  
- std::unordered\_map — Stores key-value pairs with fast access by hashing the key.  
- std::unordered\_multimap — Allows duplicate keys with fast access through hashing.  
  
# Container Adapters  
- std::stack — LIFO stack (Last-In-First-Out).  
- std::queue — FIFO queue (First-In-First-Out).  
- std::priority\_queue — Max-heap by default; stores elements in priority order.  
  
---  
  
 2. Iterators  
Iterators are used to traverse elements in containers. They act like pointers for elements.  
  
- begin() and end() — Begin and end of a container.  
- rbegin() and rend() — Reverse begin and end.  
- cbegin(), cend(), crbegin(), crend() — Const iterators.  
  
# Iterator Categories  
- Input Iterator — For read-only traversal.  
- Output Iterator — For write-only traversal.  
- Forward Iterator — For single-pass, read/write traversal.  
- Bidirectional Iterator — For forward/backward traversal.  
- Random Access Iterator — For direct access to elements (e.g., std::vector).  
  
---  
  
 3. Algorithms  
STL provides algorithms for various operations on containers. Algorithms generally operate on iterators rather than on containers directly.  
  
- Non-Modifying Algorithms — find, count, equal, search, mismatch, etc.  
- Modifying Algorithms — copy, fill, transform, swap, replace, etc.  
- Sorting and Related Algorithms — sort, stable\_sort, partial\_sort, nth\_element, partition, etc.  
- Numeric Algorithms — accumulate, inner\_product, adjacent\_difference, partial\_sum.  
- Set Algorithms — set\_union, set\_intersection, set\_difference, set\_symmetric\_difference.  
  
---  
  
 4. Functors and Function Objects  
Functors are classes that define the operator(), enabling them to be used like functions. STL provides several predefined function objects in <functional>:  
  
- Arithmetic Functors — std::plus, std::minus, std::multiplies, std::divides, etc.  
- Relational Functors — std::equal\_to, std::not\_equal\_to, std::greater, std::less, etc.  
- Logical Functors — std::logical\_and, std::logical\_or, std::logical\_not.  
  
---  
  
 5. Utilities  
Utilities offer various helper functions and types.  
  
- std::pair — Holds a pair of values.  
- std::tuple — Holds a fixed-size collection of values of heterogeneous types (C++11).  
- std::optional — Represents optional values (C++17).  
- std::variant — Holds a value that may be one of several types (C++17).  
- std::any — Type-safe container for single values of any type (C++17).  
  
---  
  
 6. Smart Pointers (C++11 and beyond)  
Smart pointers manage dynamic memory and automatically handle deallocation.  
  
- std::unique\_ptr — Exclusive ownership of a dynamically allocated object.  
- std::shared\_ptr — Shared ownership, uses reference counting.  
- std::weak\_ptr — Weak reference to an object managed by std::shared\_ptr, avoids circular references.  
  
---  
  
 7. Other Utilities  
Additional helper classes and functions to facilitate various programming tasks:  
  
- std::move — Moves an object (used for transfer of ownership).  
- std::forward — Perfect forwarding.  
- std::reference\_wrapper — Wraps references for use in STL containers.  
- std::chrono — Time utilities for measuring and manipulating time durations and points (C++11).  
- std::ratio — Compile-time rational arithmetic support (C++11).  
- std::thread, std::mutex — Threading and concurrency utilities (C++11).  
- std::filesystem — Manipulates file systems and paths (C++17).  
  
---  
  
 8. Random Number Library (C++11)  
Random number generation utilities:  
  
- Random Engines — std::default\_random\_engine, std::mt19937, std::ranlux24, etc.  
- Distributions — std::uniform\_int\_distribution, std::normal\_distribution, std::bernoulli\_distribution, etc.  
  
---  
  
These components cover the essential STL features available in C++, and each provides a foundation for efficient, reusable, and reliable code.