STLIntroduction

January 18, 2026

1 STL - Standard Template Library

1.1 External Resources

- YouTube Video - https://youtu.be/_-lxEMtPaQA
- YouTube Podcast - https://youtu.be/2pysE4rZNHI
- C++ Reference for STL: https://en.cppreference.com/w/cpp/container.html
- NotebookLM learning materials - https://notebooklm.google.com/notebook/bfc775dc-bb80-453d-b984-3ee2a2d4e902

1.2 Overview

- The Standard Template Library (STL) is a powerful set of C++ template classes to provide general-purpose classes and functions with templates that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks.
- The STL provides several benefits, including:
  - Reusability: You can use pre-written code for common data structures and algorithms.
  - Efficiency: STL implementations are often optimized for performance.
  - Flexibility: Templates allow you to create generic and reusable code that works with any data type.
- The STL consists of four components:
  1. Algorithms: Functions for searching, sorting, and manipulating data.
  2. Containers: Data structures like vectors, lists, and maps to store collections of data.
  3. Iterators: Objects that allow you to traverse and access elements in containers.
  4. Function Objects (Functors): Objects that can be used as functions, often to customize algorithms.
- The STL is widely used in C++ programming and is an essential part of modern C++ development.

1.3 STL Containers Overview

- https://www.cppreference.com/w/cpp/container.html
- Containers are data structures that store collections of objects. The STL provides several types of containers, each with its own characteristics and use cases:
- there are four primary types of STL containers:

1.3.1 Sequence Containers

- vector, list, deque, and array

- **vector**: A dynamic array that can resize itself. It provides fast random access to elements and is suitable for storing a collection of items that may change in size.
- **list**: A doubly linked list that allows for efficient insertion and deletion of elements at any position. It is suitable for scenarios where frequent modifications to the collection are required.
- **forward_list**: A singly linked list that allows for efficient insertion and deletion from anywhere in the container. It is more memory efficient than a doubly linked list but does not support backward traversal.
- **deque**: A double-ended queue that allows for fast insertion and deletion of elements at both ends. It is useful for scenarios where you need to add or remove elements from both the front and back of the collection.
- **array**: A fixed-size array that provides fast access to elements. It is suitable for scenarios where the size of the collection is known at compile time and does not change.

1.3.2 Associative Containers

- set, map, multiset, and multimap
- Associative containers implement sorted data structures that can be quickly searched (O(log n) complexity).
- **set**: An ordered collection of unique elements. It provides fast lookup, insertion, and deletion operations.
- **map**: An associative container that stores key-value pairs. It provides fast lookup, insertion, and deletion operations based on keys.
- **multiset**: Similar to a set, but allows for duplicate elements.
- **multimap**: Similar to a map, but allows for duplicate keys.

1.3.3 Unordered Associative Containers

- Unordered associative containers implement unsorted (hashed) data structures that can be quickly searched (O(1) average, O(n) worst-case complexity).
- keys are hashed into indices of a hash table to allow for fast access to individual elements based on their keys.
- unordered_set, unordered_map, unordered_multiset, and unordered_multimap
- **unordered_set**: An unordered collection of unique elements. It provides fast average-time lookup, insertion, and deletion operations.
- **unordered_map**: An unordered associative container that stores key-value pairs. It provides fast average-time lookup, insertion, and deletion operations based on keys.
- **unordered_multiset**: Similar to an unordered set, but allows for duplicate elements.
- **unordered_multimap**: Similar to an unordered map, but allows for duplicate keys

1.3.4 Container Adapters

- stack, queue, and priority_queue
- Container adapters provide a restricted interface to the underlying container.
- **stack**: A last-in, first-out (LIFO) data structure that allows for pushing and popping elements from the top of the stack.
- **queue**: A first-in, first-out (FIFO) data structure that allows for adding elements to the back and removing elements from the front.
- **priority_queue**: A data structure that allows for adding elements with priorities and removing the highest-priority element first (max priority queue).

- Each container type has its own strengths and weaknesses, and the choice of which container to use depends on the specific requirements of your application, such as the need for fast access, insertion, deletion, or ordering of elements.